

THIRD EDITION

PYTHON CRASH COURSE

A HANDS-ON, PROJECT-BASED
INTRODUCTION TO PROGRAMMING

ERIC MATTHES



Contenido en detalle

1. [Elogios para Curso acelerado de Python](#)
2. [Página del título](#)
3. [Derechos de autor](#)
4. [Dedicatoria](#)
5. [Sobre el autor](#)
6. [Prefacio a la tercera edición](#)
7. [Agradecimientos](#)
8. [Introducción](#)
 - a. [¿A quién va dirigido este libro?](#)
 - b. [¿Qué puedes esperar aprender?](#)
 - c. [Recursos en línea](#)
 - d. [¿Por qué Python?](#)
9. [Parte I: Conceptos básicos](#)
 - a. [Capítulo 1: Primeros pasos](#)
 - i. [Configurar tu entorno de programación](#)
 1. [Versiones de Python](#)
 2. [Ejecutar fragmentos de código Python](#)
 3. [Acerca del editor de código VS](#)
 - ii. [Python en diferentes sistemas operativos](#)
 1. [Python en Windows](#)
 2. [Python en macOS](#)
 3. [Python en Linux](#)
 - iii. [Ejecutar un programa Hola Mundo](#)
 1. [Instalar la extensión de Python para VS Code](#)
 2. [Ejecutar hola_mundo.py](#)
 - iv. [Solución de problemas](#)
 - v. [Ejecutar programas Python desde un terminal](#)
 1. [En Windows](#)
 2. [En macOS y Linux](#)
 3. [Exercise 1-1: python.org](#)
 4. [Exercise 1-2: Hello World Typos](#)
 5. [Exercise 1-3: Infinite Skills](#)
 - vi. [Resumen](#)
 - b. [Capítulo 2: Variables y tipos de datos simples](#)
 - i. [Qué ocurre realmente cuando ejecutas hola_mundo.py](#)
 - ii. [Variables](#)
 1. [Nombrar y utilizar variables](#)
 2. [Evitar errores de nombre al utilizar variables](#)

3. Las variables son etiquetas
 4. *Exercise 2-1: Simple Message*
 5. *Exercise 2-2: Simple Messages*
 - iii. Cadenas
 1. Cambiar mayúsculas y minúsculas en una cadena con métodos
 2. Uso de variables en cadenas
 3. Añadir espacios en blanco a las cadenas con tabuladores o nuevas líneas
 4. Eliminar espacios en blanco
 5. Eliminar prefijos
 6. Evitar errores de sintaxis con cadenas
 7. *Exercise 2-3: Personal Message*
 8. *Exercise 2-4: Name Cases*
 9. *Exercise 2-5: Famous Quote*
 10. *Exercise 2-6: Famous Quote 2*
 11. *Exercise 2-7: Stripping Names*
 12. *Exercise 2-8: File Extensions*
 - iv. Números
 1. Enteros
 2. Flotantes
 3. Enteros y flotantes
 4. Subrayados en números
 5. Asignación múltiple
 6. Constantes
 7. *Exercise 2-9: Number Eight*
 8. *Exercise 2-10: Favorite Number*
 - v. Comentarios
 1. ¿Cómo se escriben los comentarios?
 2. ¿Qué tipo de comentarios debes escribir?
 3. *Exercise 2-11: Adding Comments*
 - vi. El Zen de Python
 1. *Exercise 2-12: Zen of Python*
 - vii. Resumen
- c. Capítulo 3: Introducción a las listas
 - i. ¿Qué es una lista?
 1. Acceso a los elementos de una lista
 2. Las posiciones índice empiezan en 0, no en 1
 3. Utilizar valores individuales de una lista
 4. *Exercise 3-1: Names*
 5. *Exercise 3-2: Greetings*
 6. *Exercise 3-3: Your Own List*

- ii. **Modificar, Añadir y Eliminar Elementos**
 - 1. Modificar elementos de una lista
 - 2. Añadir elementos a una lista
 - 3. Eliminar elementos de una lista
 - 4. *Exercise 3-4: Guest List*
 - 5. *Exercise 3-5: Changing Guest List*
 - 6. *Exercise 3-6: More Guests*
 - 7. *Exercise 3-7: Shrinking Guest List*
- iii. **Organizar una Lista**
 - 1. Ordenar una Lista Permanentemente con el Método `sort()`
 - 2. Ordenar una lista temporalmente con la función `sorted()`
 - 3. Imprimir una lista en orden inverso
 - 4. Determinar la longitud de una lista
 - 5. *Exercise 3-8: Seeing the World*
 - 6. *Exercise 3-9: Dinner Guests*
 - 7. *Exercise 3-10: Every Function*
- iv. **Evitar Errores de Índice al Trabajar con Listas**
 - 1. *Exercise 3-11: Intentional Error*
- v. **Resumen**
- d. **Capítulo 4: Trabajar con listas**
 - i. **Recorrer una lista entera en bucle**
 - 1. Un vistazo más de cerca a los bucles
 - 2. Hacer más trabajo dentro de un bucle `for`
 - 3. Hacer algo después de un bucle `for`
 - ii. **Evitar errores de sangría**
 - 1. Olvidar la sangría
 - 2. Olvidar Sangrar Líneas Adicionales
 - 3. Sangrar Innecesariamente
 - 4. Sangrar innecesariamente después del bucle
 - 5. Olvidar los dos puntos
 - 6. *Exercise 4-1: Pizzas*
 - 7. *Exercise 4-2: Animals*
 - iii. **Hacer listas numéricas**
 - 1. Uso de la función `range()`
 - 2. Utilizar `range()` para hacer una lista de números
 - 3. Estadísticas sencillas con una lista de números
 - 4. Comprensión de listas
 - 5. *Exercise 4-3: Counting to Twenty*
 - 6. *Exercise 4-4: One Million*
 - 7. *Exercise 4-5: Summing a Million*

8. *Exercise 4-6: Odd Numbers*
 9. *Exercise 4-7: Threes*
 10. *Exercise 4-8: Cubes*
 11. *Exercise 4-9: Cube Comprehension*
 - iv. Trabajar con parte de una lista
 1. Cortar una lista
 2. Recorrer una porción
 3. Copiar una lista
 4. *Exercise 4-10: Slices*
 5. *Exercise 4-11: My Pizzas, Your Pizzas*
 6. *Exercise 4-12: More Loops*
 - v. Tuplas
 1. Definir una tupla
 2. Recorrer todos los valores de una tupla
 3. Escribir sobre una tupla
 4. *Exercise 4-13: Buffet*
 - vi. Estilizar tu código
 1. La Guía de Estilo
 2. Sangría
 3. Longitud de línea
 4. Líneas en blanco
 5. Otras directrices de estilo
 6. *Exercise 4-14: PEP 8*
 7. *Exercise 4-15: Code Review*
 - vii. Resumen
- e. Capítulo 5: Sentencias if
 - i. Un ejemplo sencillo
 - ii. Pruebas condicionales
 1. Comprobación de igualdad
 2. Ignorar el caso al comprobar la igualdad
 3. Comprobación de la desigualdad
 4. Comparaciones numéricas
 5. Comprobación de condiciones múltiples
 6. Comprobar si un valor está en una lista
 7. Comprobar si un valor no está en una lista
 8. Expresiones booleanas
 9. *Exercise 5-1: Conditional Tests*
 10. *Exercise 5-2: More Conditional Tests*
 - iii. Sentencias if
 1. Sentencias if simples

2. Sentencias if-else
 3. La cadena if-elif-else
 4. Uso de varios bloques elif
 5. Omitir el bloque else
 6. Probar varias condiciones
 7. *Exercise 5-3: Alien Colors #1*
 8. *Exercise 5-4: Alien Colors #2*
 9. *Exercise 5-5: Alien Colors #3*
 10. *Exercise 5-6: Stages of Life*
 11. *Exercise 5-7: Favorite Fruit*
 - iv. Uso de sentencias if con listas
 1. Comprobación de elementos especiales
 2. Comprobar que una lista no está vacía
 3. Utilizar varias listas
 4. *Exercise 5-8: Hello Admin*
 5. *Exercise 5-9: No Users*
 6. *Exercise 5-10: Checking Usernames*
 7. *Exercise 5-11: Ordinal Numbers*
 - v. Estilo de las sentencias if
 1. *Exercise 5-12: Styling if Statements*
 2. *Exercise 5-13: Your Ideas*
 - vi. Resumen
- f. Capítulo 6: Diccionarios
- i. Un diccionario sencillo
 - ii. Trabajar con diccionarios
 1. Acceder a los valores de un diccionario
 2. Añadir nuevos pares clave-valor
 3. Empezar con un Diccionario Vacío
 4. Modificar Valores de un Diccionario
 5. Eliminar Pares Clave-Valor
 6. Un Diccionario de Objetos Similares
 7. Utilizar get() para acceder a los valores
 8. *Exercise 6-1: Person*
 9. *Exercise 6-2: Favorite Numbers*
 10. *Exercise 6-3: Glossary*
 - iii. Recorrer un diccionario
 1. Recorrer todos los pares clave-valor
 2. Recorrer todas las claves de un diccionario
 3. Recorrer las claves de un diccionario en un orden determinado
 4. Recorrer todos los valores de un diccionario

5. *Exercise 6-4: Glossary 2*
 6. *Exercise 6-5: Rivers*
 7. *Exercise 6-6: Polling*
 - iv. **Anidamiento**
 1. Una lista de diccionarios
 2. Una lista en un diccionario
 3. Un diccionario en un diccionario
 4. *Exercise 6-7: People*
 5. *Exercise 6-8: Pets*
 6. *Exercise 6-9: Favorite Places*
 7. *Exercise 6-10: Favorite Numbers*
 8. *Exercise 6-11: Cities*
 9. *Exercise 6-12: Extensions*
 - v. **Resumen**
 - g. **Capítulo 7: Entrada del usuario y bucles while**
 - i. **Cómo funciona la función input()**
 1. Escribir instrucciones claras
 2. Uso de int() para aceptar entradas numéricas
 3. El operador módulo
 4. *Exercise 7-1: Rental Car*
 5. *Exercise 7-2: Restaurant Seating*
 6. *Exercise 7-3: Multiples of Ten*
 - ii. **Introducción a los bucles while**
 1. El bucle while en acción
 2. Dejar que el usuario elija cuándo salir
 3. Utilizar un indicador
 4. Usar break para salir de un bucle
 5. Usar continue en un bucle
 6. Evitar bucles infinitos
 7. *Exercise 7-4: Pizza Toppings*
 8. *Exercise 7-5: Movie Tickets*
 9. *Exercise 7-6: Three Exits*
 10. *Exercise 7-7: Infinity*
 - iii. **Utilizar un bucle while con listas y diccionarios**
 1. Mover elementos de una lista a otra
 2. Eliminar todas las instancias de determinados valores de una lista
 3. Rellenar un diccionario con datos introducidos por el usuario
 4. *Exercise 7-8: Deli*
 5. *Exercise 7-9: No Pastrami*
 6. *Exercise 7-10: Dream Vacation*

- iv. Resumen
- h. Capítulo 8: Funciones
 - i. Definir una función
 - 1. Pasar información a una función
 - 2. Argumentos y parámetros
 - 3. *Exercise 8-1: Message*
 - 4. *Exercise 8-2: Favorite Book*
 - ii. Pasar argumentos
 - 1. Argumentos Posicionales
 - 2. Argumentos de palabra clave
 - 3. Valores por defecto
 - 4. Llamadas a funciones equivalentes
 - 5. Evitar errores en los argumentos
 - 6. *Exercise 8-3: T-Shirt*
 - 7. *Exercise 8-4: Large Shirts*
 - 8. *Exercise 8-5: Cities*
 - iii. Valores de Retorno
 - 1. Devolver un Valor Simple
 - 2. Convertir un argumento en opcional
 - 3. Devolución de un diccionario
 - 4. Utilizar una función con un bucle while
 - 5. *Exercise 8-6: City Names*
 - 6. *Exercise 8-7: Album*
 - 7. *Exercise 8-8: User Albums*
 - iv. Pasar una lista
 - 1. Modificar una lista en una función
 - 2. Evitar que una función modifique una lista
 - 3. *Exercise 8-9: Messages*
 - 4. *Exercise 8-10: Sending Messages*
 - 5. *Exercise 8-11: Archived Messages*
 - v. Pasar un Número Arbitrario de Argumentos
 - 1. Mezclar argumentos posicionales y arbitrarios
 - 2. Uso de argumentos arbitrarios con palabras clave
 - 3. *Exercise 8-12: Sandwiches*
 - 4. *Exercise 8-13: User Profile*
 - 5. *Exercise 8-14: Cars*
 - vi. Almacenar tus funciones en módulos
 - 1. Importar un módulo entero
 - 2. Importar funciones específicas
 - 3. Uso de `as` para asignar un alias a una función

- 4. Utilizar `as` para asignar un alias a un módulo
 - 5. Importar todas las funciones de un módulo
 - vii. Estilizar funciones
 - 1. *Exercise 8-15: Printing Models*
 - 2. *Exercise 8-16: Imports*
 - 3. *Exercise 8-17: Styling Functions*
 - viii. Resumen
- i. Capítulo 9: Clases
 - i. Crear y utilizar una clase
 - 1. Creación de la clase `Perro`
 - 2. El método `__init__()`
 - 3. Crear una instancia a partir de una clase
 - 4. *Exercise 9-1: Restaurant*
 - 5. *Exercise 9-2: Three Restaurants*
 - 6. *Exercise 9-3: Users*
 - ii. Trabajar con clases e instancias
 - 1. La clase `Coche`
 - 2. Establecer el valor por defecto de un atributo
 - 3. Modificar los valores de un atributo
 - 4. *Exercise 9-4: Number Served*
 - 5. *Exercise 9-5: Login Attempts*
 - iii. Herencia
 - 1. El método `__init__()` de una clase hija
 - 2. Definición de atributos y métodos de la clase hija
 - 3. Sustitución de métodos de la clase padre
 - 4. Instancias como atributos
 - 5. Modelado de objetos del mundo real
 - 6. *Exercise 9-6: Ice Cream Stand*
 - 7. *Exercise 9-7: Admin*
 - 8. *Exercise 9-8: Privileges*
 - 9. *Exercise 9-9: Battery Upgrade*
 - iv. Importar clases
 - 1. Importar una sola clase
 - 2. Almacenar varias clases en un módulo
 - 3. Importar varias clases de un módulo
 - 4. Importar un módulo entero
 - 5. Importar todas las clases de un módulo
 - 6. Importar un módulo a otro módulo
 - 7. Utilizar alias
 - 8. Encontrar tu propio flujo de trabajo

- 9. *Exercise 9-10: Imported Restaurant*
 - 10. *Exercise 9-11: Imported Admin*
 - 11. *Exercise 9-12: Multiple Modules*
 - v. La biblioteca estándar de Python
 - 1. *Exercise 9-13: Dice*
 - 2. *Exercise 9-14: Lottery*
 - 3. *Exercise 9-15: Lottery Analysis*
 - 4. *Exercise 9-16: Python Module of the Week*
 - vi. Clases de Estilo
 - vii. Resumen
- j. Capítulo 10: Archivos y excepciones
 - i. Leer de un archivo
 - 1. Lectura del contenido de un archivo
 - 2. Rutas de archivos relativas y absolutas
 - 3. Acceder a las líneas de un archivo
 - 4. Trabajar con el contenido de un archivo
 - 5. Ficheros Grandes: Un millón de dígitos
 - 6. ¿Está tu cumpleaños contenido en Pi?
 - 7. *Exercise 10-1: Learning Python*
 - 8. *Exercise 10-2: Learning C*
 - 9. *Exercise 10-3: Simpler Code*
 - ii. Escribir en un fichero
 - 1. Escribir una sola línea
 - 2. Escribir varias líneas
 - 3. *Exercise 10-4: Guest*
 - 4. *Exercise 10-5: Guest Book*
 - iii. Excepciones
 - 1. Manejo de la excepción ZeroDivisionError
 - 2. Uso de bloques try-except
 - 3. Uso de excepciones para evitar fallos
 - 4. El bloque else
 - 5. Manejo de la Excepción FileNotFoundError
 - 6. Analizar texto
 - 7. Trabajar con varios archivos
 - 8. Fallos silenciosos
 - 9. Decidir de qué errores informar
 - 10. *Exercise 10-6: Addition*
 - 11. *Exercise 10-7: Addition Calculator*
 - 12. *Exercise 10-8: Cats and Dogs*
 - 13. *Exercise 10-9: Silent Cats and Dogs*

14. *Exercise 10-10: Common Words*
 - iv. Almacenamiento de datos
 1. Utilizar `json.dumps()` y `json.loads()`
 2. Guardar y leer datos generados por el usuario
 3. Refactorización de
 4. *Exercise 10-11: Favorite Number*
 5. *Exercise 10-12: Favorite Number Remembered*
 6. *Exercise 10-13: User Dictionary*
 7. *Exercise 10-14: Verify User*
 - v. Resumen
- k. Capítulo 11: Probar tu código
 - i. Instalar pytest con pip
 1. Actualizar pip
 2. Instalar pytest
 - ii. Probar una función
 1. Pruebas unitarias y casos de prueba
 2. Una prueba superada
 3. Ejecutar una prueba
 4. Una prueba fallida
 5. Respuesta a una prueba fallida
 6. Añadir nuevas pruebas
 7. *Exercise 11-1: City, Country*
 8. *Exercise 11-2: Population*
 - iii. Probar una clase
 1. Variedad de afirmaciones
 2. Una clase para probar
 3. Probar la clase `EncuestaAnónima`
 4. Uso de dispositivos
 5. *Exercise 11-3: Employee*
 - iv. Resumen
10. Parte II: Proyectos
 - a. Capítulo 12: Un barco que dispara balas
 - i. Planificar tu proyecto
 - ii. Instalar Pygame
 - iii. Iniciar el proyecto de juego
 1. Crear una ventana Pygame y responder a las entradas del usuario
 2. Control de la velocidad de fotogramas
 3. Configurar el color de fondo
 4. Crear una clase de configuración
 - iv. Añadir la imagen de la nave

1. Creación de la Clase Nave
 2. Dibujar la Nave en la Pantalla
 - v. Refactorización: Los métodos `_check_events()` y `_update_screen()`
 1. Método `_check_events()`
 2. Método `_update_screen()`
 3. *Exercise 12-1: Blue Sky*
 4. *Exercise 12-2: Game Character*
 - vi. Pilotar la nave
 1. Respuesta a la pulsación de una tecla
 2. Permitir el movimiento continuo
 3. Moverse a izquierda y derecha
 4. Ajustar la velocidad de la nave
 5. Limitar el alcance de la nave
 6. Refactorización de `_check_events()`
 7. Pulsar Q para salir
 8. Ejecutar el juego a pantalla completa
 - vii. Resumen rápido
 1. `alien_invasion.py`
 2. `settings.py`
 3. `ship.py`
 4. *Exercise 12-3: Pygame Documentation*
 5. *Exercise 12-4: Rocket*
 6. *Exercise 12-5: Keys*
 - viii. Disparar balas
 1. Añadir la configuración de las balas
 2. Crear la Clase de Bala
 3. Almacenar Balas en un Grupo
 4. Disparar Balas
 5. Borrar balas antiguas
 6. Limitar el número de balas
 7. Creación del método `_update_bullets()`
 8. *Exercise 12-6: Sideways Shooter*
 - ix. Resumen
- b. Capítulo 13: ¡Alienígenas!
 - i. Revisión del proyecto
 - ii. Creación del primer alienígena
 1. Creación de la Clase Alien
 2. Crear una Instancia del Extraterrestre
 - iii. Creación de la Flota Alienígena
 1. Creación de una Fila de Extraterrestres

2. Refactorización de `_create_fleet()`
 3. Añadir filas
 4. *Exercise 13-1: Stars*
 5. *Exercise 13-2: Better Stars*
 - iv. Mover la Flota
 1. Mover los Alienígenas a la Derecha
 2. Crear ajustes para la dirección de la flota
 3. Comprobar si un Alien ha tocado el borde
 4. Soltar la Flota y Cambiar de Dirección
 5. *Exercise 13-3: Raindrops*
 6. *Exercise 13-4: Steady Rain*
 - v. Disparar a los Alienígenas
 1. Detectar colisiones de balas
 2. Hacer balas más grandes para probarlas
 3. Repoblar la flota
 4. Acelerar las balas
 5. Refactorización de `_update_bullets()`
 6. *Exercise 13-5: Sideways Shooter Part 2*
 - vi. Finalizar el juego
 1. Detectar colisiones entre naves alienígenas
 2. Respuesta a las colisiones de naves alienígenas
 3. Alienígenas que llegan al fondo de la pantalla
 4. Fin del Juego
 5. Identificar cuándo deben ejecutarse las partes del juego
 6. *Exercise 13-6: Game Over*
 - vii. Resumen
- c. Capítulo 14: Puntuación
 - i. Añadir el Botón Jugar
 1. Crear una clase de botón
 2. Dibujar el Botón en la Pantalla
 3. Iniciar el juego
 4. Reiniciar el juego
 5. Desactivar el Botón Reproducir
 6. Ocultar el cursor del ratón
 7. *Exercise 14-1: Press P to Play*
 8. *Exercise 14-2: Target Practice*
 - ii. Subir de nivel
 1. Modificar los ajustes de velocidad
 2. Restablecer la velocidad
 3. *Exercise 14-3: Challenging Target Practice*

- 4. *Exercise 14-4: Difficulty Levels*
 - iii. Puntuación
 - 1. Mostrar la puntuación
 - 2. Crear un marcador
 - 3. Actualizar la puntuación al derribar aliens
 - 4. Reiniciar el marcador
 - 5. Asegurarse de puntuar todos los impactos
 - 6. Aumentar el valor de los puntos
 - 7. Redondear la puntuación
 - 8. Puntuaciones altas
 - 9. Mostrar el nivel
 - 10. Mostrar el número de naves
 - 11. *Exercise 14-5: All-Time High Score*
 - 12. *Exercise 14-6: Refactoring*
 - 13. *Exercise 14-7: Expanding the Game*
 - 14. *Exercise 14-8: Sideways Shooter, Final Version*
 - iv. Resumen
- d. Capítulo 15: Generación de datos
 - i. Instalar Matplotlib
 - ii. Trazar un gráfico lineal simple
 - 1. Cambiar el tipo de etiqueta y el grosor de la línea
 - 2. Corregir el trazado
 - 3. Uso de estilos incorporados
 - 4. Trazado y estilo de puntos individuales con scatter()
 - 5. Trazado de una serie de puntos con scatter()
 - 6. Cálculo automático de datos
 - 7. Personalizar las etiquetas de los puntos
 - 8. Definir colores personalizados
 - 9. Uso de un mapa de colores
 - 10. Guardar automáticamente tus trazados
 - 11. *Exercise 15-1: Cubes*
 - 12. *Exercise 15-2: Colored Cubes*
 - iii. Paseos Aleatorios
 - 1. Creación de la Clase RandomWalk
 - 2. Elección de Direcciones
 - 3. Trazado del Recorrido Aleatorio
 - 4. Generar Múltiples Paseos Aleatorios
 - 5. Estilizar el Paseo
 - 6. *Exercise 15-3: Molecular Motion*
 - 7. *Exercise 15-4: Modified Random Walks*

8. *Exercise 15-5: Refactoring*
- iv. **Tirar Dados con Plotly**
 1. Instalar Plotly
 2. Crear la clase Dado
 3. Lanzar el dado
 4. Analizar los resultados
 5. Hacer un histograma
 6. Personalizar el gráfico
 7. Lanzar dos dados
 8. Otras personalizaciones
 9. Tirar dados de diferentes tamaños
 10. Guardar cifras
 11. *Exercise 15-6: Two D8s*
 12. *Exercise 15-7: Three Dice*
 13. *Exercise 15-8: Multiplication*
 14. *Exercise 15-9: Die Comprehensions*
 15. *Exercise 15-10: Practicing with Both Libraries*
- v. **Resumen**
- e. **Capítulo 16: Descargar datos**
 - i. **El formato de archivo CSV**
 1. Análisis de las cabeceras del archivo CSV
 2. Imprimir las cabeceras y sus posiciones
 3. Extracción y lectura de datos
 4. Trazar datos en un gráfico de temperatura
 5. El módulo datetime
 6. Trazar fechas
 7. Trazar un marco temporal más largo
 8. Trazar una segunda serie de datos
 9. Sombrear un área del gráfico
 10. Comprobación de errores
 11. Descargar tus propios datos
 12. *Exercise 16-1: Sitka Rainfall*
 13. *Exercise 16-2: Sitka–Death Valley Comparison*
 14. *Exercise 16-3: San Francisco*
 15. *Exercise 16-4: Automatic Indexes*
 16. *Exercise 16-5: Explore*
 - ii. **Cartografía de Conjuntos de Datos Globales: Formato GeoJSON**
 1. Descarga de datos de terremotos
 2. Examinar los Datos GeoJSON
 3. Hacer una lista de todos los terremotos

4. [Extraer Magnitudes](#)
 5. [Extraer datos de localización](#)
 6. [Construir un mapa mundial](#)
 7. [Representación de Magnitudes](#)
 8. [Personalizar los colores de los marcadores](#)
 9. [Otras escalas de color](#)
 10. [Añadir texto flotante](#)
 11. [Exercise 16-6: Refactoring](#)
 12. [Exercise 16-7: Automated Title](#)
 13. [Exercise 16-8: Recent Earthquakes](#)
 14. [Exercise 16-9: World Fires](#)
 - iii. [Resumen](#)
- f. [Capítulo 17: Trabajar con APIs](#)
 - i. [Utilizar una API](#)
 1. [Git y GitHub](#)
 2. [Solicitar datos mediante una llamada a la API](#)
 3. [Instalación de solicitudes](#)
 4. [Procesar una respuesta de la API](#)
 5. [Trabajar con el Diccionario de Respuestas](#)
 6. [Resumiendo los principales repositorios](#)
 7. [Supervisar los Límites de Velocidad de la API](#)
 - ii. [Visualizar repositorios con Plotly](#)
 1. [Estilizar el gráfico](#)
 2. [Añadir información sobre herramientas personalizadas](#)
 3. [Añadir enlaces clicables](#)
 4. [Personalizar los colores de los marcadores](#)
 5. [Más sobre Plotly y la API de GitHub](#)
 - iii. [La API de Hacker News](#)
 1. [Exercise 17-1: Other Languages](#)
 2. [Exercise 17-2: Active Discussions](#)
 3. [Exercise 17-3: Testing python_repos.py](#)
 4. [Exercise 17-4: Further Exploration](#)
 - iv. [Resumen](#)
- g. [Capítulo 18: Primeros pasos con Django](#)
 - i. [Crear un proyecto](#)
 1. [Escribir una especificación](#)
 2. [Crear un entorno virtual](#)
 3. [Activar el entorno virtual](#)
 4. [Instalar Django](#)
 5. [Crear un Proyecto en Django](#)

6. Creación de la Base de Datos
 7. Visualización del Proyecto
 8. *Exercise 18-1: New Projects*
 - ii. Iniciar una aplicación
 1. Definición de Modelos
 2. Activar Modelos
 3. El sitio de administración de Django
 4. Definir el Modelo de Entrada
 5. Migrar el Modelo de Entrada
 6. Registro de la entrada en el sitio de administración
 7. El Shell Django
 8. *Exercise 18-2: Short Entries*
 9. *Exercise 18-3: The Django API*
 10. *Exercise 18-4: Pizzeria*
 - iii. Creación de Páginas: La Página de Inicio del Registro de Aprendizaje
 1. Mapear una URL
 2. Escribir una vista
 3. Escribir una plantilla
 4. *Exercise 18-5: Meal Planner*
 5. *Exercise 18-6: Pizzeria Home Page*
 - iv. Crear páginas adicionales
 1. Herencia de Plantillas
 2. La Página de Temas
 3. Páginas de Temas Individuales
 4. *Exercise 18-7: Template Documentation*
 5. *Exercise 18-8: Pizzeria Pages*
 - v. Resumen
- h. Capítulo 19: Cuentas de Usuario
 - i. Permitir a los Usuarios Introducir Datos
 1. Añadir Temas Nuevos
 2. Añadir Nuevas Entradas
 3. Editar Entradas
 4. *Exercise 19-1: Blog*
 - ii. Configurar cuentas de usuario
 1. La App de cuentas
 2. La página de inicio de sesión
 3. Cerrar sesión
 4. Página de registro
 5. *Exercise 19-2: Blog Accounts*
 - iii. Permitir que los usuarios sean propietarios de sus datos

1. [Restringir el acceso con @login_required](#)
 2. [Conectar Datos a Determinados Usuarios](#)
 3. [Restringir el Acceso a los Temas a los Usuarios Apropriados](#)
 4. [Proteger los Temas de un Usuario](#)
 5. [Proteger la página edit_entry](#)
 6. [Asociar Temas Nuevos al Usuario Actual](#)
 7. [Exercise 19-3: Refactoring](#)
 8. [Exercise 19-4: Protecting new_entry](#)
 9. [Exercise 19-5: Protected Blog](#)
- iv. [Resumen](#)
- i. [Capítulo 20: Estilización e implementación de una aplicación](#)
 - i. [Registro de aprendizaje del estilo](#)
 1. [La aplicación django-bootstrap5](#)
 2. [Uso de Bootstrap para dar estilo al Registro de aprendizaje](#)
 3. [Modificar base.xhtmll](#)
 4. [Estilizar la página de inicio con un jumbotrón](#)
 5. [Estilizar la página de inicio de sesión](#)
 6. [Estilizar la Página de Temas](#)
 7. [Estilizar las entradas de la página de temas](#)
 8. [Exercise 20-1: Other Forms](#)
 9. [Exercise 20-2: Stylish Blog](#)
 - ii. [Desplegar el Registro de Aprendizaje](#)
 1. [Crear una cuenta Platform.sh](#)
 2. [Instalar la CLI de Platform.sh](#)
 3. [Instalación de platformshconfig](#)
 4. [Crear un archivo requirements.txt](#)
 5. [Requisitos adicionales para el despliegue](#)
 6. [Añadir archivos de configuración](#)
 7. [Modificar settings.py para Platform.sh](#)
 8. [Utilizar Git para rastrear los archivos del proyecto](#)
 9. [Crear un proyecto en Platform.sh](#)
 10. [Enviar a Plataforma.sh](#)
 11. [Ver el proyecto activo](#)
 12. [Perfeccionar el despliegue de Platform.sh](#)
 13. [Crear páginas de error personalizadas](#)
 14. [Desarrollo en curso](#)
 15. [Eliminar un proyecto en Platform.sh](#)
 16. [Exercise 20-3: Live Blog](#)
 17. [Exercise 20-4: Extended Learning Log](#)
- iii. [Resumen](#)

11. [Apéndice A: Instalación y resolución de problemas](#)
 - a. [Python en Windows](#)
 - i. [Usar py en lugar de python](#)
 - ii. [Volver a ejecutar el instalador](#)
 - b. [Python en macOS](#)
 - i. [Instalar accidentalmente la versión de Apple de Python](#)
 - ii. [Python 2 en versiones antiguas de macOS](#)
 - c. [Python en Linux](#)
 - i. [Usar la instalación por defecto de Python](#)
 - ii. [Instalar la última versión de Python](#)
 - d. [Comprobar qué versión de Python estás utilizando](#)
 - e. [Palabras clave y funciones incorporadas de Python](#)
 - i. [Palabras clave de Python](#)
 - ii. [Funciones incorporadas de Python](#)
12. [Apéndice B: Editores de texto e IDEs](#)
 - a. [Trabajar eficazmente con VS Code](#)
 - i. [Configuración de VS Code](#)
 - ii. [Atajos de VS Code](#)
 - b. [Otros editores de texto e IDEs](#)
 - i. [IDLE](#)
 - ii. [Geany](#)
 - iii. [Sublime Text](#)
 - iv. [Emacs y Vim](#)
 - v. [PyCharm](#)
 - vi. [Cuadernos Jupyter](#)
13. [Apéndice C: Obtener ayuda](#)
 - a. [Primeros pasos](#)
 - i. [Inténtalo de nuevo](#)
 - ii. [Tómate un descanso](#)
 - iii. [Consulta los recursos de este libro](#)
 - b. [Búsqueda en línea](#)
 - i. [Desbordamiento de pila](#)
 - ii. [Documentación oficial de Python](#)
 - iii. [Documentación Oficial de la Biblioteca](#)
 - iv. [r/learnpython](#)
 - v. [Entradas de blog](#)
 - c. [Discord](#)
 - d. [Slack](#)
14. [Apéndice D: Utilizar Git para el control de versiones](#)
 - a. [Instalar Git](#)

- i. Configurar Git
 - b. Crear un proyecto
 - c. Ignorar archivos
 - d. Inicializar un repositorio
 - e. Comprobar el estado
 - f. Añadir archivos al repositorio
 - g. Hacer un commit
 - h. Comprobación del registro
 - i. La segunda confirmación
 - j. Abandonar cambios
 - k. Comprobar confirmaciones anteriores
 - l. Borrar el repositorio
15. Apéndice E: Solución de problemas en las implantaciones
- a. Comprender las implantaciones
 - b. Solución de problemas básicos
 - i. Sigue las sugerencias en pantalla
 - ii. Lee la salida del registro
 - c. Solución de problemas específicos del SO
 - i. Despliegue desde Windows
 - ii. Despliegue desde macOS
 - iii. Despliegue desde Linux
 - d. Otros métodos de implantación
16. Índice

Lista de tablas

- 1. Tabla 11-1: Declaraciones de aserción utilizadas habitualmente en las pruebas
- 2. Tabla 16-1: Argumentos de formato de fecha y hora del módulo `datetime`

Lista de ilustraciones

- 1. Figura 1-1: Asegúrate de seleccionar la casilla de verificación *Add Python to PATH*.
- 2. Figura 12-1: El barco para *Alien Invasion*
- 3. Figura 12-2: *Alien Invasion* con la nave en la parte inferior central de la pantalla
- 4. Figura 12-3: La nave después de disparar una serie de balas
- 5. Figura 13-1: El alienígena que utilizaremos para construir la flota
- 6. Figura 13-2: Aparece el primer alienígena.
- 7. Figura 13-3: La primera fila de alienígenas
- 8. Figura 13-4: Aparece la flota completa.
- 9. Figura 13-5: ¡Podemos disparar a los alienígenas!
- 10. Figura 13-6: Las balas extra potentes hacen que algunos aspectos del juego sean más fáciles de probar.
- 11. Figura 14-1: Aparece un botón Jugar cuando el juego está inactivo.

12. Figura 14-2: La puntuación aparece en la esquina superior derecha de la pantalla.
13. Figura 14-3: Puntuación redondeada con separadores de comas.
14. Figura 14-4: La puntuación más alta aparece en la parte superior central de la pantalla.
15. Figura 14-5: El nivel actual aparece justo debajo de la puntuación actual.
16. Figura 14-6: El sistema completo de puntuación de *Alien Invasion*
17. Figura 15-1: Uno de los gráficos más sencillos que puedes hacer en Matplotlib
18. Figura 15-2: El gráfico es ahora mucho más fácil de leer.
19. Figura 15-3: Ahora los datos se trazan correctamente.
20. Figura 15-4: El estilo seaborn incorporado
21. Figura 15-5: Trazado de un único punto
22. Figura 15-6: Gráfico de dispersión con varios puntos
23. Figura 15-7: Python puede trazar 1.000 puntos tan fácilmente como 5 puntos.
24. Figura 15-8: Un gráfico utilizando el mapa de colores Blues
25. Figura 15-9: Un recorrido aleatorio con 5.000 puntos
26. Figura 15-10: Un paseo aleatorio coloreado con el mapa de colores Blues
27. Figura 15-11: Un paseo con 50.000 puntos
28. Figura 15-12: Gráfico inicial producido por Plotly Express
29. Figura 15-13: Un gráfico de barras sencillo creado con Plotly
30. Figura 15-14: Resultados simulados de lanzar dos dados de seis caras 1.000 veces
31. Figura 15-15: Los resultados de lanzar un dado de seis caras y un dado de diez caras 50.000 veces
32. Figura 16-1: Gráfico lineal que muestra las temperaturas máximas diarias de julio de 2021 en Sitka, Alaska
33. Figura 16-2: El gráfico tiene más sentido, ahora que tiene fechas en el eje x .
34. Figura 16-3: Un año de datos
35. Figura 16-4: Dos series de datos en el mismo gráfico
36. Figura 16-5: La región entre los dos conjuntos de datos está sombreada.
37. Figura 16-6: Temperaturas máximas y mínimas diarias del Valle de la Muerte
38. Figura 16-7: Un mapa sencillo que muestra dónde se produjeron todos los terremotos de las últimas 24 horas
39. Figura 16-8: El mapa muestra ahora la magnitud de todos los terremotos de los últimos 30 días.
40. Figura 16-9: En los terremotos de los últimos 30 días, el color y el tamaño se utilizan para representar la magnitud de cada terremoto.
41. Figura 16-10: El texto hover incluye ahora un resumen de cada terremoto.
42. Figura 17-1: Los proyectos Python con más estrellas en GitHub
43. Figura 17-2: Se ha añadido un título al gráfico principal, y también a cada eje.
44. Figura 17-3: Al pasar el ratón por encima de una barra se muestra el propietario y la descripción del proyecto.
45. Figura 18-1: Hasta ahora todo funciona.

46. [Figura 18-2: El sitio de administración con Topic incluido](#)
47. [Figura 18-3: La página de inicio del Registro de Aprendizaje](#)
48. [Figura 18-4: La página de temas](#)
49. [Figura 18-5: La página detallada de un tema, que muestra todas las entradas de un tema](#)
50. [Figura 19-1: Página para añadir un nuevo tema](#)
51. [Figura 19-2: La página new_entry](#)
52. [Figura 19-3: Cada entrada tiene ahora un enlace para editarla.](#)
53. [Figura 19-4: La página de inicio de sesión](#)
54. [Figura 20-1: Página de inicio del Registro de Aprendizaje utilizando Bootstrap](#)
55. [Figura 20-2: La página de inicio de sesión con estilo Bootstrap](#)
56. [Figura 20-3: La página de temas con estilo Bootstrap](#)

Lista de listados

Guía

1. [Portada](#)
2. [Portada](#)
3. [Dedicatoria](#)
4. [Prefacio a la Tercera Edición](#)
5. [Introducción](#)
6. [Parte I: Conceptos básicos](#)
7. [Capítulo 1: Primeros pasos](#)
8. [Empieza a leer](#)
9. [Capítulo 2: Variables y tipos de datos simples](#)
10. [Capítulo 3: Introducción a las listas](#)
11. [Capítulo 4: Trabajar con listas](#)
12. [Capítulo 5: Sentencias if](#)
13. [Capítulo 6: Diccionarios](#)
14. [Capítulo 7: Entrada del usuario y bucles while](#)
15. [Capítulo 8: Funciones](#)
16. [Capítulo 9: Clases](#)
17. [Capítulo 10: Archivos y excepciones](#)
18. [Capítulo 11: Probar tu código](#)
19. [Parte II: Proyectos](#)
20. [Capítulo 12: Una nave que dispara balas](#)
21. [Capítulo 13: ¡Alienígenas!](#)
22. [Capítulo 14: Puntuación](#)
23. [Capítulo 15: Generar datos](#)
24. [Capítulo 16: Descarga de datos](#)
25. [Capítulo 17: Trabajar con API](#)

26. Capítulo 18: Primeros pasos con Django
27. Capítulo 19: Cuentas de usuario
28. Capítulo 20: Estilizar y desplegar una aplicación
29. Apéndice A: Instalación y resolución de problemas
30. Apéndice B: Editores de texto e IDEs
31. Apéndice C: Obtención de ayuda
32. Apéndice D: Uso de Git para el control de versiones
33. Apéndice E: Resolución de problemas de implantación
34. Índice

Páginas

1. i
2. iii
3. iv
4. v
5. vii
6. xxvii
7. xxviii
8. xxix
9. xxxi
10. xxxii
11. xxxiii
12. xxxiv
13. xxxv
14. xxxvi
15. 1
16. 2
17. 3
18. 4
19. 5
20. 6
21. 7
22. 8
23. 9
24. 10
25. 11
26. 12
27. 13
28. 15
29. 16
30. 17

31.	18
32.	19
33.	20
34.	21
35.	22
36.	23
37.	24
38.	25
39.	26
40.	27
41.	28
42.	29
43.	30
44.	31
45.	32
46.	33
47.	34
48.	35
49.	36
50.	37
51.	38
52.	39
53.	40
54.	41
55.	42
56.	43
57.	44
58.	45
59.	46
60.	47
61.	49
62.	50
63.	51
64.	52
65.	53
66.	54
67.	55
68.	56
69.	57
70.	58

71.	59
72.	60
73.	61
74.	62
75.	63
76.	64
77.	65
78.	66
79.	67
80.	68
81.	69
82.	70
83.	71
84.	72
85.	73
86.	74
87.	75
88.	76
89.	77
90.	78
91.	79
92.	80
93.	81
94.	82
95.	83
96.	84
97.	85
98.	86
99.	87
100.	88
101.	89
102.	91
103.	92
104.	93
105.	94
106.	95
107.	96
108.	97
109.	98
110.	99

111.100
112.101
113.102
114.103
115.104
116.105
117.106
118.107
119.108
120.109
121.110
122.111
123.112
124.113
125.114
126.115
127.116
128.117
129.118
130.119
131.120
132.121
133.122
134.123
135.124
136.125
137.126
138.127
139.129
140.130
141.131
142.132
143.133
144.134
145.135
146.136
147.137
148.138
149.139
150.140

151.141
152.142
153.143
154.144
155.145
156.146
157.147
158.148
159.149
160.150
161.151
162.152
163.153
164.154
165.155
166.157
167.158
168.159
169.160
170.161
171.162
172.163
173.164
174.165
175.166
176.167
177.168
178.169
179.170
180.171
181.172
182.173
183.174
184.175
185.176
186.177
187.178
188.179
189.180
190.181

191.183
192.184
193.185
194.186
195.187
196.188
197.189
198.190
199.191
200.192
201.193
202.194
203.195
204.196
205.197
206.198
207.199
208.200
209.201
210.202
211.203
212.204
213.205
214.206
215.207
216.209
217.210
218.211
219.212
220.213
221.214
222.215
223.216
224.217
225.218
226.219
227.220
228.221
229.222
230.223

231.225
232.226
233.227
234.228
235.229
236.230
237.231
238.232
239.233
240.234
241.235
242.236
243.237
244.238
245.239
246.240
247.241
248.242
249.243
250.244
251.245
252.246
253.247
254.248
255.249
256.250
257.251
258.252
259.253
260.255
261.256
262.257
263.258
264.259
265.260
266.261
267.262
268.263
269.264
270.265

271.266
272.267
273.268
274.269
275.270
276.271
277.272
278.273
279.274
280.275
281.276
282.277
283.278
284.279
285.280
286.281
287.282
288.283
289.284
290.285
291.286
292.287
293.288
294.289
295.290
296.291
297.292
298.293
299.294
300.295
301.296
302.297
303.298
304.299
305.301
306.302
307.303
308.304
309.305
310.306

311.307
312.308
313.309
314.310
315.311
316.312
317.313
318.314
319.315
320.316
321.317
322.318
323.319
324.320
325.321
326.322
327.323
328.324
329.325
330.326
331.327
332.328
333.329
334.330
335.331
336.332
337.333
338.334
339.335
340.336
341.337
342.338
343.339
344.340
345.341
346.342
347.343
348.344
349.345
350.346

351.347
352.348
353.349
354.350
355.351
356.352
357.353
358.355
359.356
360.357
361.358
362.359
363.360
364.361
365.362
366.363
367.364
368.365
369.366
370.367
371.368
372.369
373.370
374.371
375.372
376.373
377.374
378.375
379.376
380.377
381.378
382.379
383.380
384.381
385.382
386.383
387.384
388.385
389.386
390.387

391.388
392.389
393.390
394.391
395.392
396.393
397.394
398.395
399.396
400.397
401.398
402.399
403.400
404.401
405.403
406.404
407.405
408.406
409.407
410.408
411.409
412.410
413.411
414.412
415.413
416.414
417.415
418.416
419.417
420.418
421.419
422.420
423.421
424.422
425.423
426.424
427.425
428.426
429.427
430.428

431.429
432.430
433.431
434.433
435.434
436.435
437.436
438.437
439.438
440.439
441.440
442.441
443.442
444.443
445.444
446.445
447.446
448.447
449.448
450.449
451.450
452.451
453.452
454.453
455.454
456.455
457.456
458.457
459.458
460.459
461.460
462.461
463.462
464.463
465.464
466.465
467.466
468.467
469.469
470.470

471.471
472.472
473.473
474.474
475.475
476.477
477.478
478.479
479.480
480.481
481.483
482.484
483.485
484.486
485.487
486.488
487.489
488.490
489.491
490.492
491.493
492.494
493.495
494.496
495.497
496.498
497.499
498.500
499.501
500.503
501.504
502.505
503.506
504.507
505.508
506.509
507.510
508.511
509.512
510.513

511.514

Elogios para Curso acelerado de Python (Praise for Python Crash Course)

Elogios para *Python Crash Course* (Praise for *Python Crash Course*)

"Ha sido interesante ver que No Starch Press produce futuros clásicos que deberían estar junto a los libros de programación más tradicionales. *Python Crash Course* es uno de esos libros".

"It has been interesting to see No Starch Press producing future classics that should be alongside the more traditional programming books. *Python Crash Course* is one of those books."

-Greg Laden, ScienceBlogs

—Greg Laden, ScienceBlogs

"Trata algunos proyectos bastante complejos y los expone de una forma coherente, lógica y amena que atrae al lector hacia el tema".

"Deals with some rather complex projects and lays them out in a consistent, logical, and pleasant manner that draws the reader into the subject."

- *Full Circle Magazine*

— *Full Circle Magazine*

"Bien presentado, con buenas explicaciones de los fragmentos de código. El libro trabaja contigo, un pequeño paso cada vez, construyendo código más complejo, explicando lo que ocurre en todo momento."

“Well presented with good explanations of the code snippets. The book works with you, one small step at a time, building more complex code, explaining what’s going on all the way.”

-Reseñas de FlickThrough

—FlickThrough Reviews

"¡Aprender Python con *Python Crash Course* fue una experiencia extremadamente positiva! Una gran elección si eres nuevo en Python".

“Learning Python with *Python Crash Course* was an extremely positive experience! A great choice if you’re new to Python.”

-Mikke va a codificar

—Mikke Goes Coding

"Hace lo que dice en la lata, y lo hace realmente bien. . . . Presenta un gran número de ejercicios útiles, así como tres proyectos desafiantes y entretenidos."

“Does what it says on the tin, and does it really well. . . . Presents a large number of useful exercises as well as three challenging and entertaining projects.”

-RealPython.com

—RealPython.com

"Una introducción rápida pero completa a la programación con Python, *Python Crash Course* es otro libro magnífico para añadir a tu biblioteca y ayudarte a dominar por fin Python."

“A fast-paced but comprehensive introduction to programming with Python, *Python Crash Course* is another superb book to add to your library and help you finally master Python.”

-TutorialEdge.net

—TutorialEdge.net

"Una opción brillante para principiantes sin experiencia en programación. Si buscas una introducción sólida y sin complicaciones a este lenguaje tan profundo, tengo que recomendarte este libro."

"A brilliant option for complete beginners without any coding experience. If you're looking for a solid, uncomplicated intro to this very deep language, I have to recommend this book."

-QuéPixel.com

—WhatPixel.com

"Contiene literalmente todo lo que necesitas saber sobre Python e incluso más".

“Contains literally everything you need to know about Python and even more.”

-FireBearStudio.com

—FireBearStudio.com

"Aunque *Python Crash Course* utiliza Python para enseñarte a programar, también enseña habilidades de programación limpias que se aplican a la mayoría de los

demás lenguajes."

"While *Python Crash Course* uses Python to teach you to code, it also teaches clean programming skills that apply to most other languages."

-Grandes Lagos Geek

—Great Lakes Geek

Título Página (Title Page)

Derechos de autor (Copyright)

CURSO ACELERADO DE PYTHON, 3ª EDICIÓN. Copyright © 2023 por Eric Matthes.

PYTHON CRASH COURSE, 3RD EDITION. Copyright © 2023 by Eric Matthes.

Todos los derechos reservados. Ninguna parte de esta obra puede ser reproducida o transmitida de ninguna forma ni por ningún medio, electrónico o mecánico, incluyendo fotocopias, grabaciones o cualquier sistema de almacenamiento o recuperación de información, sin el permiso previo por escrito del propietario del copyright y del editor.

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher.

Primera edición

First printing

26 25 24 23 22 1 2 3 4 5

ISBN-13: 978-1-7185-0270-3 (impreso)

ISBN-13: 978-1-7185-0271-0 (ebook)

ISBN-13: 978-1-7185-0270-3 (print)

ISBN-13: 978-1-7185-0271-0 (ebook)

Editor: William Pollock

Directora editorial: Jill Franklin

Editora de Producción: Jennifer Kepler

Editora de Desarrollo: Eva Morrow

Ilustrador de portada: Josh Ellingson
Diseño interior: Octopod Studios
Revisor técnico: Kenneth Love
Corrector: Doug McNair
Compositor: Jeff Lytle, Happenstance Type-O-Rama
Corrector de pruebas: Scout Festa

Publisher: William Pollock
Managing Editor: Jill Franklin
Production Editor: Jennifer Kepler
Developmental Editor: Eva Morrow
Cover Illustrator: Josh Ellingson
Interior Design: Octopod Studios
Technical Reviewer: Kenneth Love
Copyeditor: Doug McNair
Compositor: Jeff Lytle, Happenstance Type-O-Rama
Proofreader: Scout Festa

Para obtener información sobre distribución, ventas al por mayor, ventas a empresas o traducciones, ponte en contacto directamente con No Starch Press, Inc. en info@nostarch.com o:

For information on distribution, bulk sales, corporate sales, or translations, please contact No Starch Press, Inc. directly at info@nostarch.com or:

No Starch Press, Inc.
245 8th Street, San Francisco, CA 94103
teléfono: 1.415.863.9900
www.nostarch.com

No Starch Press, Inc.
245 8th Street, San Francisco, CA 94103
phone: 1.415.863.9900
www.nostarch.com

The Library of Congress has catalogued the first edition as follows:

The Library of Congress has catalogued the first edition as follows:

Matthes, Eric, 1972-

Curso acelerado de Python : una introducción práctica a la programación basada en proyectos / por Eric Matthes.

páginas cm

Incluye índice.

Resumen: "Una introducción a la programación en Python basada en proyectos, con ejercicios. Cubre conceptos generales de programación, fundamentos de Python y resolución de problemas. Incluye tres proyectos: cómo crear un videojuego sencillo, utilizar técnicas de visualización de datos para hacer gráficos y diagramas, y construir una aplicación web interactiva"-- Proporcionado por el editor.

ISBN 978-1-59327-603-4 -- ISBN 1-59327-603-6

1. Python (Lenguaje de programación informática). Python (Lenguaje de programación informática) I. Título.

QA76.73.P98M38 2015

005.13'3--dc23

2015018135

Matthes, Eric, 1972-

Python crash course : a hands-on, project-based introduction to programming / by Eric Matthes.

pages cm

Includes index.

Summary: "A project-based introduction to programming in Python, with exercises. Covers general programming concepts, Python fundamentals, and problem solving. Includes three projects - how to create a simple video game, use data visualization techniques to make graphs and charts, and build an interactive web application"-- Provided by publisher.

ISBN 978-1-59327-603-4 -- ISBN 1-59327-603-6

1. Python (Computer program language) I. Title.

QA76.73.P98M38 2015

005.13'3--dc23

2015018135

No Starch Press y el logotipo de No Starch Press son marcas registradas de No Starch Press, Inc. Otros nombres de productos y empresas aquí mencionados pueden ser marcas registradas de sus respectivos propietarios. En lugar de utilizar un símbolo de marca registrada cada vez que aparece un nombre de marca registrada, utilizamos los nombres

sólo de forma editorial y en beneficio del propietario de la marca registrada, sin intención de infringir la marca registrada.

No Starch Press and the No Starch Press logo are registered trademarks of No Starch Press, Inc. Other product and company names mentioned herein may be the trademarks of their respective owners. Rather than use a trademark symbol with every occurrence of a trademarked name, we are using the names only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

La información de este libro se distribuye "tal cual", sin garantía. Aunque se han tomado todas las precauciones en la preparación de esta obra, ni el autor ni No Starch Press, Inc. tendrán responsabilidad alguna ante ninguna persona o entidad con respecto a cualquier pérdida o daño causado o supuestamente causado directa o indirectamente por la información contenida en ella.

The information in this book is distributed on an "As Is" basis, without warranty. While every precaution has been taken in the preparation of this work, neither the author nor No Starch Press, Inc. shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in it.

Dedicatoria (Dedication)

Para mi padre, que siempre sacaba tiempo para responder a mis preguntas sobre programación, y para Ever, que está empezando a hacerme sus preguntas

For my father, who always made time to answer my questions about programming, and for Ever, who is just beginning to ask me his questions

Sobre el autor (About the Author)

Sobre el autor (*About the Author*)

Eric Matthes fue profesor de matemáticas y ciencias en un instituto durante 25 años, y daba clases introductorias de Python siempre que encontraba la forma de encajarlas en el plan de estudios. Ahora Eric es escritor y programador a tiempo completo, y participa en varios proyectos de código abierto. Sus proyectos tienen objetivos muy diversos, desde ayudar a predecir la actividad de los corrimientos de tierra en regiones montañosas hasta simplificar el proceso de despliegue de proyectos Django. Cuando no está escribiendo o programando, le gusta escalar montañas y pasar tiempo con su familia.

Eric Matthes was a high school math and science teacher for 25 years, and he taught introductory Python classes whenever he could find a way to fit them into the curriculum. Eric is a full-time writer and programmer now, and he is involved in a number of open source projects. His projects have a diverse range of goals, from helping predict landslide activity in mountainous regions to simplifying the process of deploying Django projects. When he's not writing or programming, he enjoys climbing mountains and spending time with his family.

Acerca del revisor técnico (*About the Technical Reviewer*)

Kenneth Love vive en el noroeste del Pacífico con su familia y sus gatos. Kenneth es programador de Python desde hace mucho tiempo, colaborador de código abierto, profesor y ponente en conferencias.

Kenneth Love lives in the Pacific Northwest with their family and cats. Kenneth is a longtime Python programmer, open source contributor, teacher, and conference speaker.

Prefacio de la Tercera Edición (Preface to the Third Edition)

Prefacio a la Tercera Edición (*Preface to the Third Edition*)

La respuesta a la primera y segunda ediciones de *Python Crash Course* ha sido abrumadoramente positiva. Hay más de un millón de ejemplares impresos, incluidas traducciones a más de 10 idiomas. He recibido cartas y correos electrónicos de lectores de tan sólo 10 años, así como de jubilados que quieren aprender a programar en su tiempo libre. *Python Crash Course* se está utilizando en colegios e institutos, y también en clases universitarias. Los estudiantes a los que se les asignan libros de texto más avanzados están utilizando *Python Crash Course* como texto complementario para sus clases y lo consideran un complemento que merece la pena. La gente lo utiliza para mejorar sus habilidades en el trabajo, cambiar de profesión y empezar a trabajar en sus propios proyectos paralelos. En resumen, la gente utiliza el libro para todos los fines que yo esperaba, y mucho más.

The response to the first and second editions of *Python Crash Course* has been overwhelmingly positive. More than one million copies are in print, including translations in over 10 languages. I've received letters and emails from readers as young as 10, as well as from retirees who want to learn to program in their free time. *Python Crash Course* is being used in middle schools and high schools, and also in college classes. Students who are assigned more advanced textbooks are using *Python Crash Course* as a companion text for their classes and finding it a worthwhile supplement. People are using it to enhance their skills on the job, change careers, and start working on their own side projects. In short, people are using the book for the full range of purposes I had hoped they would, and much more.

La oportunidad de escribir una tercera edición de *Python Crash Course* ha sido muy agradable. Aunque Python es un lenguaje maduro, sigue evolucionando como todos los lenguajes. Mi principal objetivo al revisar el libro es que siga siendo un curso de introducción a Python bien elaborado. Al leer este libro, aprenderás todo lo que necesitas para empezar a trabajar en tus propios proyectos, y además construirás una base sólida para todo tu aprendizaje futuro. He actualizado algunas secciones para reflejar formas más nuevas y sencillas de hacer las cosas en Python. También he aclarado algunas secciones en las que ciertos detalles del lenguaje no se presentaban con toda la precisión posible. Todos los proyectos se han actualizado completamente utilizando bibliotecas populares y bien mantenidas que puedes utilizar con confianza para construir tus propios proyectos.

The opportunity to write a third edition of *Python Crash Course* has been thoroughly enjoyable. Although Python is a mature language, it continues to evolve as every language does. My main goal in revising the book is to keep it a well-curated introductory Python course. By reading this book, you'll learn everything you need to start working on your own projects, and you'll build a solid foundation for all of your future learning as well. I've updated some sections to reflect newer, simpler ways of doing things in Python. I've also clarified some sections where certain details of the language were not presented as accurately as they could have been. All the projects have been completely updated using popular, well-maintained libraries that you can confidently use to build your own projects.

A continuación se resumen los cambios específicos que se han introducido en la tercera edición:

The following is a summary of specific changes that have been made in the third edition:

Listado en español:

- El Capítulo 1 incluye ahora el editor de texto VS Code, que es popular entre los programadores principiantes y profesionales y funciona bien en todos los sistemas operativos.
- El Capítulo 2 incluye los nuevos métodos `removeprefix()` y `removesuffix()`, que son útiles cuando se trabaja con archivos y URLs. Este capítulo también incluye los nuevos mensajes de error mejorados de Python, que proporcionan información mucho más específica para ayudarte a solucionar los problemas de tu código cuando algo va mal.
- El capítulo 10 utiliza el módulo `pathlib` para trabajar con archivos. Se trata de un enfoque mucho más sencillo para leer y escribir en archivos.
- El Capítulo 11 utiliza `pytest` para escribir pruebas automatizadas del código que escribes. La biblioteca `pytest` se ha convertido en la herramienta estándar de la industria para escribir pruebas en Python. Es lo suficientemente amigable como para utilizarla en tus primeras pruebas, y si sigues una carrera como programador de Python, también la utilizarás en entornos profesionales.
- El proyecto *Alien Invasion* de los Capítulos 12-14 incluye un ajuste para controlar la velocidad de fotogramas, que hace que el juego funcione de forma más consistente en diferentes sistemas operativos. Se utiliza un enfoque más sencillo para construir la flota de alienígenas, y también se ha limpiado la organización general del proyecto.
- Los proyectos de visualización de datos de los Capítulos 15-17 utilizan las funciones más recientes de Matplotlib y Plotly. Las visualizaciones de Matplotlib presentan ajustes de estilo actualizados. El proyecto de paseo aleatorio tiene una pequeña mejora que aumenta la precisión de los trazados, lo que significa que verás

surgir una mayor variedad de patrones cada vez que generes un nuevo paseo. Todos los proyectos que incluyen Plotly utilizan ahora el módulo Plotly Express, que te permite generar tus visualizaciones iniciales con sólo unas pocas líneas de código. Puedes explorar fácilmente una variedad de visualizaciones antes de comprometerte con un tipo de gráfico, y luego centrarte en refinar elementos individuales de ese gráfico.

- El proyecto Learning Log de los Capítulos 18-20 se ha creado con la última versión de Django y se le ha dado estilo con la última versión de Bootstrap. Algunas partes del proyecto se han renombrado para que sea más fácil seguir la organización general del proyecto. El proyecto está ahora desplegado en Platform.sh, un moderno servicio de alojamiento para proyectos Django. El proceso de despliegue se controla mediante archivos de configuración YAML, que te proporcionan un gran control sobre cómo se despliega tu proyecto. Este enfoque es coherente con la forma en que los programadores profesionales despliegan los proyectos Django modernos.
- El Apéndice A se ha actualizado completamente para recomendar las mejores prácticas actuales para instalar Python en los principales sistemas operativos. El Apéndice B incluye instrucciones detalladas para configurar VS Code, y breves descripciones de la mayoría de los principales editores de texto e IDE de uso actual. El Apéndice C dirige a los lectores a varios de los recursos en línea más populares para obtener ayuda en . El Apéndice D sigue ofreciendo un minicurso intensivo sobre el uso de Git para el control de versiones. El Apéndice E es totalmente nuevo para la tercera edición. Incluso con un buen conjunto de instrucciones para desplegar las aplicaciones que creas, hay muchas cosas que pueden salir mal. Este apéndice ofrece una guía detallada de resolución de problemas que puedes utilizar cuando el proceso de despliegue no funcione a la primera.
- El índice se ha actualizado a fondo para que puedas utilizar *Python Crash Course* como referencia para todos tus futuros proyectos con Python.

Listing in English:

- Chapter 1 now features the text editor VS Code, which is popular among beginner and professional programmers and works well on all operating systems.
- Chapter 2 includes the new methods `removeprefix()` and `removesuffix()`, which are helpful when working with files and URLs. This chapter also features Python's newly improved error messages, which provide much more specific information to help you troubleshoot your code when something goes wrong.
- Chapter 10 uses the `pathlib` module for working with files. This is a much simpler approach to reading from and writing to files.
- Chapter 11 uses `pytest` to write automated tests for the code you write. The `pytest` library has become the industry standard tool for writing tests in Python. It's friendly enough to use for your first tests, and if you pursue a career as a Python programmer, you'll use it in professional settings as well.
- The *Alien Invasion* project in Chapters 12–14 includes a setting to control the frame rate, which makes the game run more consistently across different operating

systems. A simpler approach is used to build the fleet of aliens, and the overall organization of the project has been cleaned up as well.

- The data visualization projects in Chapters 15–17 use the most recent features of Matplotlib and Plotly. The Matplotlib visualizations feature updated style settings. The random walk project has a small improvement that increases the accuracy of the plots, which means you'll see a wider variety of patterns emerge each time you generate a new walk. All the projects featuring Plotly now use the Plotly Express module, which lets you generate your initial visualizations with just a few lines of code. You can easily explore a variety of visualizations before committing to one kind of plot, and then focus on refining individual elements of that plot.
- The Learning Log project in Chapters 18–20 is built using the latest version of Django and styled using the latest version of Bootstrap. Some parts of the project have been renamed to make it easier to follow the overall organization of the project. The project is now deployed to Platform.sh, a modern hosting service for Django projects. The deployment process is controlled by YAML configuration files, which give you a great deal of control over how your project is deployed. This approach is consistent with how professional programmers deploy modern Django projects.
- Appendix A has been fully updated to recommend current best practices for installing Python on all major operating systems. Appendix B includes detailed instructions for setting up VS Code, and brief descriptions of most of the major text editors and IDEs in current use. Appendix C directs readers to several of the most popular online resources for getting help. Appendix D continues to offer a mini crash course in using Git for version control. Appendix E is brand new for the third edition. Even with a good set of instructions for deploying the apps you create, there are many things that can go wrong. This appendix offers a detailed troubleshooting guide that you can use when the deployment process doesn't work on the first try.
- The index has been thoroughly updated to allow you to use *Python Crash Course* as a reference for all of your future Python projects.

¡Gracias por leer *Python Crash Course* ! Si tienes algún comentario o pregunta, no dudes en ponerte en contacto conmigo; soy [@ehmatthes](#) en Twitter.

Thank you for reading *Python Crash Course* ! If you have any feedback or questions, please feel free to get in touch; I am [@ehmatthes](#) on Twitter.

Agradecimientos (Acknowledgments)

Agradecimientos (Acknowledgments)

Este libro no habría sido posible sin el maravilloso y extremadamente profesional equipo de No Starch Press. Bill Pollock me invitó a escribir un libro introductorio, y agradezco profundamente esa original oferta. Liz Chadwick ha trabajado en las tres ediciones, y el libro es mejor gracias a su continua participación. Eva Morrow aportó una mirada nueva a esta nueva edición, y sus ideas también han mejorado el libro. Agradezco la orientación de Doug McNair para utilizar una gramática adecuada, sin llegar a ser excesivamente formal. Jennifer Kepler ha supervisado el trabajo de producción, que convierte mis numerosos archivos en un producto final pulido.

This book would not have been possible without the wonderful and extremely professional staff at No Starch Press. Bill Pollock invited me to write an introductory book, and I deeply appreciate that original offer. Liz Chadwick has worked on all three editions, and the book is better because of her ongoing involvement. Eva Morrow brought fresh eyes to this new edition, and her insights have improved the book as well. I appreciate Doug McNair's guidance in using proper grammar, without becoming overly formal. Jennifer Kepler has supervised the production work, which turns my many files into a polished final product.

Hay muchas personas en No Starch Press que han ayudado a que este libro sea un éxito, pero con las que no he tenido la oportunidad de trabajar directamente. No Starch tiene un equipo de marketing fantástico, que va más allá de la mera venta de libros; se aseguran de que los lectores encuentren los libros que probablemente les vayan bien y les ayuden a alcanzar sus objetivos. No Starch también tiene un sólido departamento de derechos en el extranjero. *Python Crash Course* ha llegado a lectores de todo el mundo, en muchos idiomas, gracias a la diligencia de este equipo. A todas estas personas con las que no he trabajado individualmente, gracias por ayudar a *Python Crash Course* a encontrar su público.

There are many people at No Starch Press who have helped make this book a success but whom I haven't had the chance to work with directly. No Starch has a fantastic marketing team, who go beyond just selling books; they make sure readers find the books that are likely to work well for them and help them reach their goals. No Starch also has a strong foreign-rights department. *Python Crash Course* has reached readers around the world, in many languages, due to the diligence of this team. To all of these people whom I haven't worked with individually, thank you for helping *Python Crash Course* find its audience.

Me gustaría dar las gracias a Kenneth Love, el revisor técnico de las tres ediciones de *Python Crash Course*. Conocí a Kenneth en la PyCon un año, y su entusiasmo por el lenguaje y la comunidad Python ha sido una fuente constante de inspiración profesional desde entonces. Kenneth, como siempre, fue más allá de la simple comprobación de hechos y revisó el libro con el objetivo de ayudar a los programadores noveles a desarrollar una sólida comprensión del lenguaje Python y de la programación en general. También se mantuvieron atentos a las áreas que funcionaban suficientemente bien en ediciones anteriores, pero que podrían mejorarse, dada la oportunidad de una reescritura completa. Dicho esto, cualquier inexactitud que quede es completamente mía.

I'd like to thank Kenneth Love, the technical reviewer for all three editions of *Python Crash Course*. I met Kenneth at PyCon one year, and their enthusiasm for the language and the Python community has been a constant source of professional inspiration ever since. Kenneth, as always, went beyond simple fact-checking and reviewed the book with the goal of helping newer programmers develop a solid understanding of the Python language and programming in general. They also kept an eye out for areas that worked well enough in previous editions but could be improved upon, given the opportunity for a full rewrite. That said, any inaccuracies that remain are completely my own.

También me gustaría expresar mi agradecimiento a todos los lectores que han compartido su experiencia de trabajo a través de *Python Crash Course*. Aprender los fundamentos de la programación puede cambiar tu perspectiva del mundo, y a veces esto tiene un profundo impacto en las personas. Es profundamente humillante escuchar estas historias, y agradezco a todos los que han compartido sus experiencias tan abiertamente.

I'd also like to express my appreciation to all the readers who have shared their experience of working through *Python Crash Course*. Learning the basics of programming can change your perspective on the world, and sometimes this has a profound impact on people. It's deeply humbling to hear these stories, and I appreciate everyone who has shared their experiences so openly.

Me gustaría dar las gracias a mi padre por iniciarme en la programación a una edad temprana y por no tener miedo de que rompiera su equipo. Me gustaría dar las gracias a mi mujer, Erin, por apoyarme y animarme a escribir este libro y por todo el trabajo que supone mantenerlo a lo largo de múltiples ediciones. También me gustaría dar las gracias a mi hijo Ever, cuya curiosidad sigue inspirándome.

I'd like to thank my father for introducing me to programming at a young age and for not being afraid that I'd break his equipment. I'd like to thank my wife, Erin, for supporting and encouraging me through the writing of this book, and through all the work that goes into maintaining it through multiple editions. I'd also like to thank my son, Ever, whose curiosity continues to inspire me.

Introducción (Introduction)

Introducción (Introduction)



Todo programador tiene una historia sobre cómo aprendió a escribir su primer programa. Yo empecé a programar de niño, cuando mi padre trabajaba para Digital Equipment Corporation, una de las empresas pioneras de la era informática moderna. Escribí mi primer programa en un ordenador kit que mi padre había montado en nuestro sótano. El ordenador no era más que una placa base desnuda conectada a un teclado sin carcasa, y su monitor era un tubo de rayos catódicos desnudo. Mi programa inicial era un simple juego de adivinar números, que tenía este aspecto:

Every programmer has a story about how they learned to write their first program. I started programming as a child, when my father was working for Digital Equipment Corporation, one of the pioneering companies of the modern computing era. I wrote my first program on a kit computer that my dad had assembled in our basement. The computer consisted of nothing more than a bare motherboard connected to a keyboard without a case, and its monitor was a bare cathode ray tube. My initial program was a simple number guessing game, which looked something like this:

I'm thinking of a number! Try to guess the number I'm thinking of: 25

Too low! Guess again: 50

Too high! Guess again: 42

That's it! Would you like to play again? (yes/no) no

Thanks for playing!

Siempre recordaré lo satisfecho que me sentí, viendo a mi familia jugar a un juego que yo había creado y que funcionaba como yo pretendía.

I'll always remember how satisfied I felt, watching my family play a game that I created and that worked as I intended it to.

Esa experiencia temprana tuvo un impacto duradero. Hay una verdadera satisfacción en construir algo con un propósito, que resuelva un problema. El software que escribo ahora satisface necesidades más importantes que las que satisfacían mis esfuerzos infantiles, pero la sensación de satisfacción que me produce crear un programa que funciona sigue siendo en gran medida la misma.

That early experience had a lasting impact. There's real satisfaction in building something with a purpose, that solves a problem. The software I write now meets more significant needs than my childhood efforts did, but the sense of satisfaction I get from creating a program that works is still largely the same.

¿A quién va dirigido este libro? (*Who Is This Book For?*)

El objetivo de este libro es ponerte al día con Python lo más rápidamente posible para que puedas crear programas que funcionen -juegos, visualizaciones de datos y aplicaciones web-, al tiempo que desarrollas una base de programación que te servirá para el resto de tu vida. *Python Crash Course* está escrito para personas de cualquier edad que nunca hayan programado en Python o que nunca hayan programado en absoluto. Este libro es para quienes quieren aprender rápidamente los fundamentos de la programación para poder centrarse en proyectos interesantes, y para quienes les gusta poner a prueba su comprensión de nuevos conceptos resolviendo problemas significativos. *Python Crash Course* también es perfecto para profesores de todos los niveles que quieran ofrecer a sus alumnos una introducción a la programación basada en proyectos. Si estás dando una clase en la universidad y quieres una introducción a Python más amigable que el texto que te han asignado, este libro también puede facilitarte la clase. Si quieres cambiar de profesión, *Python Crash Course* puede ayudarte a hacer la transición a una carrera más satisfactoria.

Ha funcionado bien para una gran variedad de lectores, con una amplia gama de objetivos.

The goal of this book is to bring you up to speed with Python as quickly as possible so you can build programs that work—games, data visualizations, and web applications—while developing a foundation in programming that will serve you well for the rest of your life. *Python Crash Course* is written for people of any age who have never programmed in Python or have never programmed at all. This book is for those who want to learn the basics of programming quickly so they can focus on interesting projects, and those who like to test their understanding of new concepts by solving meaningful problems. *Python Crash Course* is also perfect for teachers at all levels who want to offer their students a project-based introduction to programming. If you're taking a college class and want a friendlier introduction to Python than the text you've been assigned, this book can make your class easier as well. If you're looking to change careers, *Python Crash Course* can help you make the transition into a more satisfying career track. It has worked well for a wide variety of readers, with a broad range of goals.

¿Qué puedes esperar aprender? (What Can You Expect to Learn?)

El propósito de este libro es convertirte en un buen programador en general y en un buen programador de Python en particular. Aprenderás de forma eficaz y adoptarás buenos hábitos a medida que adquieras una base sólida en conceptos generales de programación. Después de abrirte camino a través de *Python Crash Course*, estarás preparado para pasar a técnicas más avanzadas de Python, y tu próximo lenguaje de programación será aún más fácil de comprender.

The purpose of this book is to make you a good programmer in general and a good Python programmer in particular. You'll learn efficiently and adopt good habits as you gain a solid foundation in general programming concepts. After working your way through *Python Crash Course*, you should be ready to move on to more advanced Python techniques, and your next programming language will be even easier to grasp.

En la Parte I de este libro, aprenderás los conceptos básicos de programación que necesitas conocer para escribir programas en Python. Estos conceptos son los mismos que aprenderías al iniciarte en casi cualquier lenguaje de programación. Conocerás los distintos tipos de datos y las formas en que puedes almacenarlos en tus programas. Construirás colecciones de datos, como listas y diccionarios, y trabajarás con esas colecciones de forma eficiente. Aprenderás a utilizar los bucles `while` y las sentencias `if` para comprobar determinadas condiciones, de modo que puedas ejecutar secciones específicas de código mientras esas condiciones sean verdaderas y ejecutar otras secciones cuando no lo sean, una técnica que te ayuda a automatizar muchos procesos.

In Part I of this book, you'll learn basic programming concepts you need to know to write Python programs. These concepts are the same as those you'd learn when starting out in almost any programming language. You'll learn about different kinds of data and the ways you can store data in your programs. You'll build collections of data, such as lists and dictionaries, and you'll work through those collections in efficient ways. You'll learn to use while loops and if statements to test for certain conditions, so you can run specific sections of code while those conditions are true and run other sections when they're not—a technique that helps you automate many processes.

Aprenderás a aceptar entradas de los usuarios para que tus programas sean interactivos, y a mantener tus programas en ejecución tanto tiempo como desee el usuario. Explorarás cómo escribir funciones que hagan reutilizables partes de tu programa, de modo que sólo tengas que escribir bloques de código que realicen determinadas acciones una vez, mientras utilizas ese código tantas veces como necesites. Luego ampliarás este concepto a comportamientos más complicados con clases, haciendo que programas bastante sencillos respondan a diversas situaciones. Aprenderás a escribir programas que manejen errores comunes con elegancia. Después de trabajar cada uno de estos conceptos básicos, escribirás una serie de programas cada vez más complejos utilizando lo que has aprendido. Por último, darás tu primer paso hacia la programación intermedia aprendiendo a escribir pruebas para tu código, de modo que puedas seguir desarrollando tus programas sin preocuparte de introducir errores. Toda la información de la Parte I te preparará para emprender proyectos más grandes y complejos.

You'll learn to accept input from users to make your programs interactive, and to keep your programs running as long as the user wants. You'll explore how to write functions that make parts of your program reusable, so you only have to write blocks of code that perform certain actions once, while using that code as many times as you need. You'll then extend this concept to more complicated behavior with classes, making fairly simple programs respond to a variety of situations. You'll learn to write programs that handle common errors gracefully. After working through each of these basic concepts, you'll write a number of increasingly complex programs using what you've learned. Finally, you'll take your first step toward intermediate programming by learning how to write tests for your code, so you can develop your programs further without worrying about introducing bugs. All the information in Part I will prepare you for taking on larger, more complex projects.

En la Parte II, aplicarás lo aprendido en la Parte I a tres proyectos. Puedes realizar cualquiera de estos proyectos o todos ellos, en el orden que mejor te convenga. En el primer proyecto, en los Capítulos 12-14, crearás un juego de disparos al estilo *Space Invaders* llamado *Alien Invasion*, que incluye varios niveles de juego cada vez más difíciles.

Cuando hayas completado este proyecto, estarás bien encaminado para poder desarrollar tus propios juegos 2D. Incluso si no aspiras a convertirte en programador de juegos, trabajar en este proyecto es una forma agradable de unir gran parte de lo que aprenderás en la Parte I.

In Part II, you'll apply what you learned in Part I to three projects. You can do any or all of these projects, in whichever order works best for you. In the first project, in Chapters 12–14 , you'll create a *Space Invaders* –style shooting game called *Alien Invasion* , which includes several increasingly difficult levels of game play. After you've completed this project, you should be well on your way to being able to develop your own 2D games. Even if you don't aspire to become a game programmer, working through this project is an enjoyable way to tie together much of what you'll learn in Part I.

El segundo proyecto, en los Capítulos 15-17 , te introduce en la visualización de datos. Los científicos de datos utilizan diversas técnicas de visualización para dar sentido a la enorme cantidad de información de que disponen. Trabajarás con conjuntos de datos que generes mediante código, conjuntos de datos que descargues de fuentes online y conjuntos de datos que tus programas descarguen automáticamente. Cuando hayas completado este proyecto, serás capaz de escribir programas que examinen grandes conjuntos de datos y hagan representaciones visuales de muchos tipos diferentes de información.

The second project, in Chapters 15–17 , introduces you to data visualization. Data scientists use a variety of visualization techniques to help make sense of the vast amount of information available to them. You'll work with datasets that you generate through code, datasets that you download from online sources, and datasets your programs download automatically. After you've completed this project, you'll be able to write programs that sift through large datasets and make visual representations of many different kinds of information.

En el tercer proyecto, en los Capítulos 18-20 , construirás una pequeña aplicación web llamada Registro de Aprendizaje. Este proyecto te permite llevar un diario organizado de la información que has aprendido sobre un tema concreto. Podrás mantener registros separados para diferentes temas y permitir que otras personas creen una cuenta e inicien sus propios diarios. También aprenderás a desplegar tu proyecto para que cualquiera pueda acceder a él en línea, desde cualquier parte del mundo.

In the third project, in Chapters 18–20 , you'll build a small web application called Learning Log. This project allows you to keep an organized journal of information you've

learned about a specific topic. You'll be able to keep separate logs for different topics and allow others to create an account and start their own journals. You'll also learn how to deploy your project so anyone can access it online, from anywhere in the world.

Recursos en línea (*Online Resources*)

No Starch Press tiene más información sobre este libro disponible en línea en <https://nostarch.com/python-crash-course-3rd-edition>.

No Starch Press has more information about this book available online at <https://nostarch.com/python-crash-course-3rd-edition>.

También mantengo un amplio conjunto de recursos complementarios en https://ehmatthes.github.io/pcc_3e. Estos recursos incluyen lo siguiente:

I also maintain an extensive set of supplementary resources at https://ehmatthes.github.io/pcc_3e. These resources include the following:

Instrucciones de configuración Las instrucciones de configuración en línea son idénticas a las del libro, pero incluyen enlaces activos en los que puedes hacer clic para seguir los distintos pasos. Si tienes algún problema de configuración, consulta este recurso.

Setup instructions The setup instructions online are identical to what's in the book, but they include active links you can click for all the different steps. If you're having any setup issues, refer to this resource.

Actualizaciones Python, como todos los lenguajes, evoluciona constantemente. Mantengo un minucioso conjunto de actualizaciones, así que si algo no funciona, comprueba aquí si han cambiado las instrucciones.

Updates Python, like all languages, is constantly evolving. I maintain a thorough set of updates, so if anything isn't working, check here to see whether instructions have changed.

Soluciones a los ejercicios Deberías dedicar bastante tiempo a intentar por tu cuenta los ejercicios de las secciones "Pruébalo tú mismo". Sin embargo, si estás atascado y no consigues avanzar, las soluciones a la mayoría de los ejercicios están en línea.

Solutions to exercises You should spend significant time on your own attempting the exercises in the "Try It Yourself" sections. However, if you're stuck and can't make any progress, solutions to most of the exercises are online.

Hojas de trucos También hay en línea un juego completo de hojas de trucos descargables para consultar rápidamente los conceptos principales.

Cheat sheets A full set of downloadable cheat sheets for quick reference to major concepts is also online.

¿Por qué Python? (Why Python?)

Todos los años me planteo si seguir utilizando Python o pasarme a otro lenguaje, quizá uno más nuevo en el mundo de la programación. Pero sigo centrándome en Python por muchas razones. Python es un lenguaje increíblemente eficiente: tus programas harán más en menos líneas de código de lo que requerirían muchos otros lenguajes. La sintaxis de Python también te ayudará a escribir código "limpio". Tu código será más fácil de leer, más fácil de depurar y más fácil de ampliar y desarrollar, en comparación con otros lenguajes.

Every year, I consider whether to continue using Python or move on to a different language, perhaps one that's newer to the programming world. But I continue to focus on Python for many reasons. Python is an incredibly efficient language: your programs will do more in fewer lines of code than many other languages would require. Python's syntax will also help you write "clean" code. Your code will be easier to read, easier to debug, and easier to extend and build upon, compared to other languages.

La gente utiliza Python con muchos fines: para hacer juegos, crear aplicaciones web, resolver problemas empresariales y desarrollar herramientas internas en todo tipo de empresas interesantes. Python también se utiliza mucho en el ámbito científico, para la investigación académica y el trabajo aplicado.

People use Python for many purposes: to make games, build web applications, solve business problems, and develop internal tools at all kinds of interesting companies. Python is also used heavily in scientific fields, for academic research and applied work.

Una de las razones más importantes por las que sigo utilizando Python es la comunidad Python, que incluye un grupo de personas increíblemente diverso y acogedor. La comunidad es esencial para los programadores porque la programación no es una actividad solitaria. La mayoría de nosotros, incluso los programadores más experimentados, necesitamos pedir consejo a otros que ya han resuelto problemas similares. Tener una comunidad bien conectada y que te apoye es fundamental para ayudarte a resolver problemas, y la comunidad Python apoya plenamente a las personas que están aprendiendo Python como su primer lenguaje de programación o que llegan a Python con conocimientos previos de otros lenguajes.

One of the most important reasons I continue to use Python is because of the Python community, which includes an incredibly diverse and welcoming group of people. Community is essential to programmers because programming isn't a solitary pursuit. Most of us, even the most experienced programmers, need to ask advice from others who have already solved similar problems. Having a well-connected and supportive community is critical to helping you solve problems, and the Python community is fully supportive of people who are learning Python as their first programming language or coming to Python with a background in other languages.

Python es un gran lenguaje para aprender, ¡así que empecemos!

Python is a great language to learn, so let's get started!

Parte I: Conceptos básicos (Part I: Basics)

Parte I Conceptos básicos (Part I Basics)

La Parte I de este libro te enseña los conceptos básicos que necesitarás para escribir programas en Python. Muchos de estos conceptos son comunes a todos los lenguajes de programación, por lo que te serán útiles a lo largo de tu vida como programador.

Part I of this book teaches you the basic concepts you'll need to write Python programs. Many of these concepts are common to all programming languages, so they'll be useful throughout your life as a programmer.

En el **Capítulo 1** instalarás Python en tu ordenador y ejecutarás tu primer programa, que imprime el mensaje *Hello world!* en la pantalla.

In **Chapter 1** you'll install Python on your computer and run your first program, which prints the message *Hello world!* to the screen.

En el **Capítulo 2** aprenderás a asignar información a variables y a trabajar con texto y valores numéricos.

In **Chapter 2** you'll learn to assign information to variables and work with text and numerical values.

Los capítulos 3 y 4 presentan las listas. Las listas pueden almacenar tanta información como quieras en un solo lugar, lo que te permite trabajar con esos datos de forma eficiente. Podrás trabajar con cientos, miles e incluso millones de valores en unas pocas líneas de código.

Chapters 3 and 4 introduce lists. Lists can store as much information as you want in one place, allowing you to work with that data efficiently. You'll be able to work with hundreds, thousands, and even millions of values in just a few lines of code.

En el **Capítulo 5** utilizarás las sentencias `if` para escribir código que responda de una manera si ciertas condiciones son verdaderas, y responda de una manera diferente si esas condiciones no son verdaderas.

In **Chapter 5** you'll use `if` statements to write code that responds one way if certain conditions are true, and responds in a different way if those conditions are not true.

El Capítulo 6 te muestra cómo utilizar los diccionarios de Python, que te permiten establecer conexiones entre distintas piezas de información. Al igual que las listas, los diccionarios pueden contener tanta información como necesites almacenar.

Chapter 6 shows you how to use Python's dictionaries, which let you make connections between different pieces of information. Like lists, dictionaries can contain as much information as you need to store.

En el **Capítulo 7** aprenderás a aceptar entradas de los usuarios para que tus programas sean interactivos. También aprenderás sobre los bucles `while`, que ejecutan bloques de código repetidamente mientras se cumplan determinadas condiciones.

In **Chapter 7** you'll learn how to accept input from users to make your programs interactive. You'll also learn about `while` loops, which run blocks of code repeatedly as long as certain conditions remain true.

En el **Capítulo 8** escribirás funciones, que son bloques de código con nombre que realizan una tarea específica y pueden ejecutarse siempre que las necesites.

In **Chapter 8** you'll write functions, which are named blocks of code that perform a specific task and can be run whenever you need them.

El Capítulo 9 presenta las clases, que te permiten modelar objetos del mundo real. Escribirás código que represente perros, gatos, personas, coches, cohetes y mucho más.

Chapter 9 introduces classes, which allow you to model real-world objects. You'll write code that represents dogs, cats, people, cars, rockets, and more.

El Capítulo 10 te muestra cómo trabajar con archivos y gestionar errores para que tus programas no se bloqueen inesperadamente. Almacenarás datos antes de que se cierre tu programa y los volverás a leer cuando el programa se ejecute de nuevo. Conocerás las excepciones de Python, que te permiten anticiparte a los errores y hacer que tus programas los manejen con elegancia.

Chapter 10 shows you how to work with files and handle errors so your programs won't crash unexpectedly. You'll store data before your program closes and read the data back in when the program runs again. You'll learn about Python's exceptions, which allow you to anticipate errors and make your programs handle those errors gracefully.

En **el Capítulo 11** aprenderás a escribir pruebas para tu código, para comprobar que tus programas funcionan como pretendes. Como resultado, podrás ampliar tus programas sin preocuparte de introducir nuevos errores. Probar tu código es una de las primeras habilidades que te ayudarán en la transición de programador principiante a programador intermedio.

In **Chapter 11** you'll learn to write tests for your code, to check that your programs work the way you intend them to. As a result, you'll be able to expand your programs without worrying about introducing new bugs. Testing your code is one of the first skills that will help you transition from beginner to intermediate programmer.

Capítulo 1: Primeros pasos (Chapter 1: Getting Started)

1

Primeros pasos (1 Getting Started)



DESCRIPCIÓN DE LA
IMAGEN (En español):

La imagen muestra una ilustración en blanco y negro de una serpiente que parece estar enrollada alrededor de una turbina de motor a reacción. La serpiente está representada con detalles cuidadosamente diseñados, destacando su textura escamosa y su expresión casi caricaturesca. La turbina está bien definida, con aspas visibles, lo que sugiere un movimiento rápido o fuerza. El fondo es simple y de color gris, haciendo que los elementos principales resalten con más claridad.

En este capítulo, ejecutarás tu primer programa Python, *hello_world.py* . En primer lugar, tendrás que comprobar si tienes instalada en tu ordenador una versión reciente de Python; si no es así, la instalarás. También instalarás un editor de texto para trabajar con tus programas Python. Los editores de texto reconocen el código Python y resaltan las secciones a medida que escribes, facilitando la comprensión de la estructura de tu código.

In this chapter, you'll run your first Python program, *hello_world.py* . First, you'll need to check whether a recent version of Python is installed on your computer; if it isn't, you'll install it. You'll also install a text editor to work with your Python programs. Text editors recognize Python code and highlight sections as you write, making it easy to understand your code's structure.

Configurar tu entorno de programación (*Setting Up Your Programming Environment*)

Python difiere ligeramente en los distintos sistemas operativos, por lo que tendrás que tener en cuenta algunas consideraciones. En las siguientes secciones, nos aseguraremos de que Python está configurado correctamente en tu sistema.

Python differs slightly on different operating systems, so you'll need to keep a few considerations in mind. In the following sections, we'll make sure Python is set up correctly on your system.

Versiones de Python (*Python Versions*)

Todo lenguaje de programación evoluciona a medida que surgen nuevas ideas y tecnologías, y los desarrolladores de Python han hecho que el lenguaje sea cada vez más versátil y potente. En el momento de escribir estas líneas, la última versión es Python 3.11, pero todo lo que aparece en este libro debería funcionar con Python 3.9 o posterior. En esta sección, averiguaremos si Python ya está instalado en tu sistema y si necesitas instalar una versión más reciente. El Apéndice A también contiene detalles adicionales sobre la instalación de la última versión de Python en cada uno de los principales sistemas operativos.

Every programming language evolves as new ideas and technologies emerge, and the developers of Python have continually made the language more versatile and powerful. As of this writing, the latest version is Python 3.11, but everything in this book should run on Python 3.9 or later. In this section, we'll find out if Python is already installed on your system and whether you need to install a newer version. Appendix A contains additional details about installing the latest version of Python on each major operating system as well.

Ejecutar fragmentos de código Python (*Running Snippets of Python Code*)

Puedes ejecutar el intérprete de Python en una ventana de terminal, lo que te permite probar fragmentos de código Python sin tener que guardar y ejecutar un programa entero.

You can run Python's interpreter in a terminal window, allowing you to try bits of Python code without having to save and run an entire program.

A lo largo de este libro, verás fragmentos de código con este aspecto:

Throughout this book, you'll see code snippets that look like this:

```
>>> print("Hello Python interpreter!")  
Hello Python interpreter!
```

La indicación de tres corchetes angulares (`>>>`), a la que nos referiremos como *Python prompt* , indica que debes utilizar la ventana del terminal. El texto en negrita es el código que debes escribir y luego ejecutar pulsando ENTER. La mayoría de los ejemplos de este libro son pequeños programas autónomos que ejecutarás desde tu editor de textos en lugar de desde el terminal, porque escribirás la mayor parte del código en el editor de textos. Pero a veces, los conceptos básicos se mostrarán en una serie de fragmentos ejecutados a través de una sesión de terminal de Python para demostrar conceptos concretos de forma más eficaz. Cuando veas tres corchetes angulares en un listado de código, estarás viendo código y salida de una sesión de terminal. Dentro de un momento intentaremos codificar en el intérprete de tu sistema.

The three angle brackets (`>>>`) prompt, which we'll refer to as a *Python prompt* , indicates that you should be using the terminal window. The bold text is the code you should type in and then execute by pressing ENTER. Most of the examples in this book are small, self-contained programs that you'll run from your text editor rather than the terminal, because you'll write most of your code in the text editor. But sometimes, basic concepts will be shown in a series of snippets run through a Python terminal session to demonstrate particular concepts more efficiently. When you see three angle brackets in a code listing, you're looking at code and output from a terminal session. We'll try coding in the interpreter on your system in a moment.

También utilizaremos un editor de texto para crear un sencillo programa llamado *Hello World!* que se ha convertido en un elemento básico para aprender a programar. Existe una larga tradición en el mundo de la programación según la cual imprimir el mensaje `Hello world!` en la pantalla como primer programa en un nuevo lenguaje te traerá buena suerte. Un programa tan sencillo tiene un propósito muy real. Si se ejecuta correctamente en tu sistema, cualquier programa Python que escribas también debería funcionar.

We'll also use a text editor to create a simple program called *Hello World!* that has become a staple of learning to program. There's a long-held tradition in the programming world that printing the message `Hello world!` to the screen as your first program in a new language will bring you good luck. Such a simple program serves a very real purpose. If it runs correctly on your system, then any Python program you write should work as well.

Acerca del editor de código VS (About the VS Code Editor)

VS Code es un potente editor de texto de calidad profesional, gratuito y apto para principiantes. *VS Code* es ideal tanto para proyectos sencillos como complejos, de modo que si te sientes cómodo utilizándolo mientras aprendes Python, podrás seguir usándolo a medida que avances hacia proyectos más grandes y complicados. *VS Code* puede instalarse en todos los sistemas operativos modernos, y es compatible con la mayoría de los lenguajes de programación, incluido Python.

VS Code is a powerful, professional-quality text editor that's free and beginner-friendly. *VS Code* is great for both simple and complex projects, so if you become comfortable using it while learning Python, you can continue using it as you progress to larger and more complicated projects. *VS Code* can be installed on all modern operating systems, and it supports most programming languages, including Python.

En el Apéndice B encontrarás información sobre otros editores de texto. Si tienes curiosidad por conocer las otras opciones, quizá quieras hojear ese apéndice en este punto de . Si quieres empezar a programar rápidamente, puedes utilizar *VS Code* para empezar. Luego puedes considerar otros editores, una vez que hayas adquirido cierta experiencia como programador. En este capítulo, te guiaré a través de la instalación de *VS Code* en tu sistema operativo.

Appendix B provides information on other text editors. If you're curious about the other options, you might want to skim that appendix at this point. If you want to begin programming quickly, you can use *VS Code* to start. Then you can consider other editors, once you've gained some experience as a programmer. In this chapter, I'll walk you through installing *VS Code* on your operating system.

Nota (Note)

Si ya tienes instalado un editor de texto y sabes cómo configurarlo para ejecutar programas Python, puedes utilizar ese editor en su lugar.

If you already have a text editor installed and you know how to configure it to run Python programs, you are welcome to use that editor instead.

Python en diferentes sistemas operativos (*Python on Different Operating Systems*)

Python es un lenguaje de programación multiplataforma, lo que significa que funciona en los principales sistemas operativos. Cualquier programa Python que escribas debería funcionar en cualquier ordenador moderno que tenga Python instalado. Sin embargo, los métodos para configurar Python en diferentes sistemas operativos varían ligeramente.

Python is a cross-platform programming language, which means it runs on all the major operating systems. Any Python program you write should run on any modern computer that has Python installed. However, the methods for setting up Python on different operating systems vary slightly.

En esta sección aprenderás a configurar Python en tu sistema. Primero comprobarás si hay una versión reciente de Python instalada en tu sistema, y la instalarás si no es así. Luego instalarás VS Code. Estos son los dos únicos pasos que son diferentes para cada sistema operativo.

In this section, you'll learn how to set up Python on your system. You'll first check whether a recent version of Python is installed on your system, and install it if it's not. Then you'll install VS Code. These are the only two steps that are different for each operating system.

En las secciones siguientes, ejecutarás *hello_world.py* y solucionarás cualquier problema que no funcione. Te guiaré a través de este proceso para cada sistema operativo, para que tengas un entorno de programación Python en el que puedas confiar.

In the sections that follow, you'll run *hello_world.py* and troubleshoot anything that doesn't work. I'll walk you through this process for each operating system, so you'll have a Python programming environment that you can rely on.

Python en Windows (Python on Windows)

Windows no suele venir con Python, así que probablemente tendrás que instalarlo y luego instalar VS Code.

Windows doesn't usually come with Python, so you'll probably need to install it and then install VS Code.

Instalar Python (Installing Python)

En primer lugar, comprueba si Python está instalado en tu sistema. Abre una ventana de comandos introduciendo `command` en el menú Inicio y haciendo clic en la aplicación **Símbolo del sistema**. En la ventana de terminal, introduce `python` en minúsculas. Si recibes como respuesta un prompt de Python (`>>>`), Python está instalado en tu sistema. Si aparece un mensaje de error indicándote que `python` no es un comando reconocido, o si se abre la tienda de Microsoft, Python no está instalado. Cierra la tienda de Microsoft si se abre; es mejor descargar un instalador oficial que utilizar la versión de Microsoft.

First, check whether Python is installed on your system. Open a command window by entering `command` into the Start menu and clicking the **Command Prompt** app. In the terminal window, enter `python` in lowercase. If you get a Python prompt (`>>>`) in response, Python is installed on your system. If you see an error message telling you that `python` is not a recognized command, or if the Microsoft store opens, Python isn't installed. Close the Microsoft store if it opens; it's better to download an official installer than to use Microsoft's version.

Si Python no está instalado en tu sistema, o si ves una versión anterior a Python 3.9, necesitas descargar un instalador de Python para Windows. Ve a <https://python.org> y pasa el ratón por encima del enlace **Descargas**. Deberías ver un botón para descargar la última versión de Python. Haz clic en el botón, que debería iniciar automáticamente la descarga del instalador correcto para tu sistema. Cuando hayas descargado el archivo, ejecuta el instalador. Asegúrate de seleccionar la opción **Añadir Python al PATH**, que facilitará la correcta configuración de tu sistema. La [Figura 1-1](#) muestra esta opción seleccionada.

If Python is not installed on your system, or if you see a version earlier than Python 3.9, you need to download a Python installer for Windows. Go to <https://python.org> and hover over the **Downloads** link. You should see a button for downloading the latest version of Python. Click the button, which should automatically start downloading the correct installer for your system. After you've downloaded the file, run the installer. Make sure you select the option **Add Python to PATH**, which will make it easier to configure your system correctly. [Figure 1-1](#) shows this option selected.



DESCRIPCIÓN DE LA IMAGEN (En español):

Instalar Python 3.11.0 (64 bits) Selecciona Instalar Ahora para instalar Python con la configuración predeterminada, o elige Personalizar para habilitar o deshabilitar funciones. Instalar Ahora C: \Usuarios\eric\AppData\Local\Programs\Python\Python311 Incluye IDLE, pip y documentación Crea accesos directos y asociaciones de archivos Personalizar instalación Elige ubicación e instalaciones Instalar lanzador para todos los usuarios (recomendado) Agregar Python 3.11 a PATH

Figura 1-1 : Asegúrate de seleccionar la casilla etiquetada Add Python to PATH

Figure 1-1 : Make sure you select the checkbox labeled Add Python to PATH.

Ejecutar Python en una sesión de terminal (Running Python in a Terminal Session)

Abre una nueva ventana de comandos e introduce `python` en minúsculas. Deberías ver un prompt de Python (`>>>`), lo que significa que Windows ha encontrado la versión de Python que acabas de instalar.

Open a new command window and enter `python` in lowercase. You should see a Python prompt (`>>>`), which means Windows has found the version of Python you just installed.

```
C:\> python
Python 3.x.x (main, Jun . . . , 13:29:14) [MSC v.1932 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Nota (Note)

Si no ves esta salida o algo similar, consulta las instrucciones de instalación más detalladas del Apéndice A .

If you don't see this output or something similar, see the more detailed setup instructions in Appendix A .

Introduce la siguiente línea en tu sesión de Python:

Enter the following line in your Python session:

```
>>> print("Hello Python interpreter!")
Hello Python interpreter!
>>>
```

Deberías ver la salida `Hello Python interpreter!` Cada vez que quieras ejecutar un fragmento de código Python, abre una ventana de comandos e inicia una sesión de terminal Python. Para cerrar la sesión de terminal, pulsa CTRL-Z y después ENTER, o introduce el

comando `exit()` .

You should see the output `Hello Python interpreter!` Anytime you want to run a snippet of Python code, open a command window and start a Python terminal session. To close the terminal session, press CTRL-Z and then press ENTER, or enter the command `exit()` .

Instalar código VS (Installing VS Code)

Puedes descargar un instalador para VS Code en <https://code.visualstudio.com> . Haz clic en el botón **Descargar para Windows** y ejecuta el instalador. Omite las secciones siguientes sobre macOS y Linux, y sigue los pasos de " Ejecutar un programa Hola Mundo " en la página 9 .

You can download an installer for VS Code at <https://code.visualstudio.com> . Click the **Download for Windows** button and run the installer. Skip the following sections about macOS and Linux, and follow the steps in " Running a Hello World Program " on page 9 .

Python en macOS (Python on macOS)

Python no está instalado por defecto en las últimas versiones de macOS, así que tendrás que instalarlo si aún no lo has hecho. En esta sección, instalarás la última versión de Python y, a continuación, instalarás VS Code y te asegurarás de que está configurado correctamente.

Python is not installed by default on the latest versions of macOS, so you'll need to install it if you haven't already done so. In this section, you'll install the latest version of Python, and then install VS Code and make sure it's configured correctly.

Nota (Note)

Python 2 se incluía en versiones anteriores de macOS, pero es una versión obsoleta que no deberías utilizar.

Python 2 was included on older versions of macOS, but it's an outdated version that you shouldn't use.

Comprobar si Python 3 está instalado (Checking Whether Python 3 Is Installed)

Abre una ventana de terminal yendo a **Aplicaciones** ▶ **Utilidades** ▶ **Terminal** . También puedes pulsar la barra espaciadora , escribir `terminal` , y pulsar INTRO. Para ver si tienes instalada una versión suficientemente reciente de Python, escribe `python3` . Lo más probable es que veas un mensaje sobre la instalación de *command line developer tools* . Es mejor instalar estas herramientas después de instalar Python, así que si aparece este mensaje, cancela la ventana emergente.

Open a terminal window by going to **Applications** ▶ **Utilities** ▶ **Terminal** . You can also press -spacebar, type `terminal` , and then press ENTER. To see if you have a recent enough version of Python installed, enter `python3` . You'll most likely see a message about installing the *command line developer tools* . It's better to install these tools after installing Python, so if this message appears, cancel the pop-up window.

Si la salida muestra que tienes instalado Python 3.9 o una versión posterior, puedes saltarte la siguiente sección y pasar a " Ejecutar Python en una sesión de terminal ". Si ves alguna versión anterior a Python 3.9, sigue las instrucciones de la siguiente sección para instalar la última versión.

If the output shows you have Python 3.9 or a later version installed, you can skip the next section and go to " Running Python in a Terminal Session ." If you see any version earlier than Python 3.9, follow the instructions in the next section to install the latest version.

Ten en cuenta que en macOS, siempre que veas el comando `python` en este libro, debes utilizar en su lugar el comando `python3` para asegurarte de que estás utilizando Python 3. En la mayoría de los sistemas macOS, el comando `python` o bien apunta a una versión obsoleta de Python que sólo deberían utilizar las herramientas internas del sistema, o bien no apunta a nada y genera un mensaje de error.

Note that on macOS, whenever you see the `python` command in this book, you need to use the `python3` command instead to make sure you're using Python 3. On most macOS

systems, the `python` command either points to an outdated version of Python that should only be used by internal system tools, or it points to nothing and generates an error message.

Instalar la última versión de Python (Installing the Latest Version of Python)

Puedes encontrar un instalador de Python para tu sistema en <https://python.org>. Pasa el ratón por encima del enlace **Descargar**, y deberías ver un botón para descargar la última versión de Python. Haz clic en el botón, que debería iniciar automáticamente la descarga del instalador correcto para tu sistema. Una vez descargado el archivo, ejecuta el instalador.

You can find a Python installer for your system at <https://python.org>. Hover over the **Download** link, and you should see a button for downloading the latest version of Python. Click the button, which should automatically start downloading the correct installer for your system. After the file downloads, run the installer.

Una vez ejecutado el instalador, debería aparecer una ventana del Finder. Haz doble clic en el archivo *Install Certificates.command*. Ejecutar este archivo te permitirá instalar más fácilmente bibliotecas adicionales que necesitarás para proyectos del mundo real, incluidos los proyectos de la segunda mitad de este libro.

After the installer runs, a Finder window should appear. Double-click the *Install Certificates.command* file. Running this file will allow you to more easily install additional libraries that you'll need for real-world projects, including the projects in the second half of this book.

Ejecutar Python en una sesión de terminal (Running Python in a Terminal Session)

Ahora puedes probar a ejecutar fragmentos de código Python abriendo una nueva ventana de terminal y escribiendo `python3` :

You can now try running snippets of Python code by opening a new terminal window and typing `python3` :

```
$ python3
Python 3.x.x (v3.11.0:eb0004c271, Jun . . . , 10:03:01)
[Clang 13.0.0 (clang-1300.0.29.30)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Este comando inicia una sesión de terminal de Python. Deberías ver un prompt de Python (`>>>`), lo que significa que macOS ha encontrado la versión de Python que acabas de instalar.

This command starts a Python terminal session. You should see a Python prompt (`>>>`), which means macOS has found the version of Python you just installed.

Introduce la siguiente línea en la sesión de terminal:

Enter the following line in the terminal session:

```
>>> print("Hello Python interpreter!")
Hello Python interpreter!
>>>
```

Deberías ver el mensaje `Hello Python interpreter!` , que debería imprimirse directamente en la ventana de terminal actual. Puedes cerrar el intérprete de Python pulsando CTRL-D o introduciendo el comando `exit()` .

You should see the message `Hello Python interpreter!` , which should print directly in the current terminal window. You can close the Python interpreter by pressing CTRL-D or by entering the command `exit()` .

Nota (Note)

En los sistemas macOS más recientes, verás un signo de porcentaje (`%`) como indicador de terminal en lugar de un signo de dólar (`$`).

On newer macOS systems, you'll see a percent sign (%) as a terminal prompt instead of a dollar sign (\$).

Instalar VS Code (Installing VS Code)

Para instalar el editor VS Code, tienes que descargar el instalador en <https://code.visualstudio.com> . Haz clic en el botón **Descargar** y, a continuación, abre una ventana **del Finder** y ve a la carpeta **Descargas** . Arrastra el instalador de **Visual Studio Code** a tu carpeta Aplicaciones y, a continuación, haz doble clic en el instalador para ejecutarlo.

To install the VS Code editor, you need to download the installer at <https://code.visualstudio.com> . Click the **Download** button, and then open a **Finder** window and go to the **Downloads** folder. Drag the **Visual Studio Code** installer to your Applications folder, then double-click the installer to run it.

Omite la siguiente sección sobre Python en Linux, y sigue los pasos de " Ejecutar un programa Hola Mundo " en la página 9 .

Skip over the following section about Python on Linux, and follow the steps in “ Running a Hello World Program ” on page 9 .

Python en Linux (Python on Linux)

Los sistemas Linux están diseñados para programar, por lo que Python ya está instalado en la mayoría de los ordenadores Linux. Las personas que escriben y mantienen Linux esperan que hagas tu propia programación en algún momento, y te animan a hacerlo. Por esta razón, hay muy poco que instalar y sólo unos pocos ajustes que cambiar para empezar a programar.

Linux systems are designed for programming, so Python is already installed on most Linux computers. The people who write and maintain Linux expect you to do your own programming at some point, and encourage you to do so. For this reason, there's very little to install and only a few settings to change to start programming.

Comprobar tu versión de Python (Checking Your Version of Python)

Abre una ventana de terminal ejecutando la aplicación Terminal en tu sistema (en Ubuntu, puedes pulsar CTRL-ALT-T). Para saber qué versión de Python está instalada, escribe `python3` con minúscula *p*. Cuando Python está instalado, este comando inicia el intérprete de Python. Deberías ver una salida indicando qué versión de Python está instalada. También deberías ver un prompt de Python (`>>>`) donde puedes empezar a introducir comandos de Python:

Open a terminal window by running the Terminal application on your system (in Ubuntu, you can press CTRL-ALT-T). To find out which version of Python is installed, enter `python3` with a lowercase *p*. When Python is installed, this command starts the Python interpreter. You should see output indicating which version of Python is installed. You should also see a Python prompt (`>>>`) where you can start entering Python commands:

```
$ python3
Python 3.10.4 (main, Apr . . . , 09:04:19) [GCC 11.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Esta salida indica que Python 3.10.4 es actualmente la versión por defecto de Python instalada en este ordenador. Cuando hayas visto esta salida, pulsa CTRL-D o escribe `exit()` para salir del indicador de Python y volver al indicador de terminal. Siempre que veas el comando `python` en este libro, introduce `python3` en su lugar.

This output indicates that Python 3.10.4 is currently the default version of Python installed on this computer. When you've seen this output, press CTRL-D or enter `exit()` to leave the Python prompt and return to a terminal prompt. Whenever you see the `python` command in this book, enter `python3` instead.

Necesitarás Python 3.9 o posterior para ejecutar el código de este libro. Si la versión de Python instalada en tu sistema es anterior a Python 3.9, o si quieres actualizarte a la última versión disponible actualmente, consulta las instrucciones del Apéndice A.

You'll need Python 3.9 or later to run the code in this book. If the Python version installed on your system is earlier than Python 3.9, or if you want to update to the latest version currently available, refer to the instructions in Appendix A.

Ejecutar Python en una sesión de terminal (Running Python in a Terminal Session)

Puedes probar a ejecutar fragmentos de código Python abriendo un terminal e introduciendo `python3` , como hiciste al comprobar tu versión. Hazlo de nuevo, y cuando tengas Python en ejecución, introduce la siguiente línea en la sesión de terminal:

You can try running snippets of Python code by opening a terminal and entering `python3` , as you did when checking your version. Do this again, and when you have Python running, enter the following line in the terminal session:

```
>>> print("Hello Python interpreter!")
Hello Python interpreter!
>>>
```

El mensaje debería imprimirse directamente en la ventana actual del terminal. Recuerda que puedes cerrar el intérprete de Python pulsando CTRL-D o introduciendo el comando `exit()` .

The message should print directly in the current terminal window. Remember that you can close the Python interpreter by pressing CTRL-D or by entering the command `exit()` .

Instalación de VS Code (Installing VS Code)

En Ubuntu Linux, puedes instalar VS Code desde el Centro de Software de Ubuntu. Haz clic en el icono de Software de Ubuntu en tu menú y busca `vscode` . Haz clic en la aplicación llamada **Visual Studio Code** (a veces denominada `code`) y, a continuación, haz clic en **Instalar** . Una vez instalada, busca `VS Code` en tu sistema e inicia la aplicación.

On Ubuntu Linux, you can install VS Code from the Ubuntu Software Center. Click the Ubuntu Software icon in your menu and search for `vscode` . Click the app called **Visual Studio Code** (sometimes called `code`), and then click **Install** . Once it's installed, search your system for `VS Code` and launch the app.

Ejecutar un programa Hola Mundo (*Running a Hello World Program*)

Con una versión reciente de Python y VS Code instalados, estás casi listo para ejecutar tu primer programa Python escrito en un editor de texto. Pero antes de hacerlo, necesitas instalar la extensión de Python para VS Code.

With a recent version of Python and VS Code installed, you're almost ready to run your first Python program written in a text editor. But before doing so, you need to install the Python extension for VS Code.

Instalar la extensión de Python para VS Code (*Installing the Python Extension for VS Code*)

VS Code funciona con muchos lenguajes de programación diferentes; para sacarle el máximo partido como programador de Python, necesitarás instalar la extensión Python. Esta extensión añade soporte para escribir, editar y ejecutar programas Python.

VS Code works with many different programming languages; to get the most out of it as a Python programmer, you'll need to install the Python extension. This extension adds support for writing, editing, and running Python programs.

Para instalar la extensión Python, haz clic en el icono Administrar, que parece un engranaje en la esquina inferior izquierda de la aplicación VS Code. En el menú que aparece, haz clic en **Extensiones**. Introduce `python` en el cuadro de búsqueda y haz clic en la extensión de **Python**. (Si ves más de una extensión llamada *Python*, elige la suministrada por Microsoft). Haz clic en **Instalar** e instala las herramientas adicionales que tu sistema necesite para completar la instalación. Si ves un mensaje indicando que necesitas instalar Python, y ya lo has hecho, puedes ignorar este mensaje.

To install the Python extension, click the Manage icon, which looks like a gear in the lower-left corner of the VS Code app. In the menu that appears, click **Extensions**. Enter `python` in the search box and click the **Python** extension. (If you see more than one extension named *Python*, choose the one supplied by Microsoft.) Click **Install** and install any additional tools that your system needs to complete the installation. If you see a message that you need to install Python, and you've already done so, you can ignore this message.

Nota (Note)

Si estás utilizando macOS y una ventana emergente te pide que instales *command line developer tools*, haz clic en **Instalar**. Es posible que aparezca un mensaje indicando que la instalación tardará demasiado tiempo, pero sólo debería tardar unos 10 ó 20 minutos con una conexión a Internet razonable.

If you're using macOS and a pop-up asks you to install the *command line developer tools*, click **Install**. You may see a message that it will take an excessively long time to install, but it should only take about 10 or 20 minutes on a reasonable internet connection.

Ejecutar `hola_mundo.py` (Running `hello_world.py`)

Antes de escribir tu primer programa, crea una carpeta llamada *python_work* en tu escritorio para tus proyectos. Es mejor utilizar letras minúsculas y guiones bajos en lugar de espacios en los nombres de archivos y carpetas, porque Python utiliza estas convenciones de nomenclatura. Puedes hacer esta carpeta en otro lugar que no sea el escritorio, pero será más fácil seguir algunos pasos posteriores si guardas la carpeta *python_work* directamente en tu escritorio.

Before you write your first program, make a folder called *python_work* on your desktop for your projects. It's best to use lowercase letters and underscores for spaces in file and folder names, because Python uses these naming conventions. You can make this folder somewhere other than the desktop, but it will be easier to follow some later steps if you save the *python_work* folder directly on your desktop.

Abre VS Code, y cierra la pestaña "**Empezar**" si todavía está abierta. Crea un nuevo archivo haciendo clic en **Archivo** ▶ **Nuevo archivo** o pulsando CTRL-N (-N en macOS). Guarda el archivo como *hello_world.py* en tu carpeta *python_work*. La extensión *.py* indica a VS Code que tu archivo está escrito en Python, y le indica cómo ejecutar el programa y resaltar el texto de forma útil.

Open VS Code, and close the **Get Started** tab if it's still open. Make a new file by clicking **File** ▶ **New File** or pressing CTRL-N (-N on macOS). Save the file as *hello_world.py* in

your *python_work* folder. The extension *.py* tells VS Code that your file is written in Python, and tells it how to run the program and highlight the text in a helpful way.

Después de guardar tu archivo, introduce la siguiente línea en el editor:

After you've saved your file, enter the following line in the editor:

hola_mundo.py

hello_world.py

```
print("Hello Python world!")
```

Para ejecutar tu programa, selecciona **Ejecutar**  **Ejecutar sin depurar** o pulsa CTRL-F5. Debería aparecer una pantalla de terminal en la parte inferior de la ventana de VS Code, mostrando la salida de tu programa:

To run your program, select **Run**  **Run Without Debugging** or press CTRL-F5. A terminal screen should appear at the bottom of the VS Code window, showing your program's output:

Hello Python world!

Es probable que veas alguna salida adicional que muestre el intérprete de Python que se utilizó para ejecutar tu programa. Si quieres simplificar la información que se muestra para que sólo veas la salida de tu programa, consulta el Apéndice B . También puedes encontrar sugerencias útiles sobre cómo utilizar VS Code de forma más eficiente en el Apéndice B .

You'll likely see some additional output showing the Python interpreter that was used to run your program. If you want to simplify the information that's displayed so you only see your program's output, see Appendix B . You can also find helpful suggestions about how to use VS Code more efficiently in Appendix B .

Si no ves esta salida, es posible que algo haya ido mal en el programa. Comprueba cada carácter de la línea que has introducido. ¿Has puesto accidentalmente `print` en mayúsculas? ¿Olvidaste una o ambas comillas o paréntesis? Los lenguajes de programación esperan una sintaxis muy específica, y si no la proporcionas, obtendrás errores. Si no consigues que el programa se ejecute, consulta las sugerencias de la siguiente sección.

If you don't see this output, something might have gone wrong in the program. Check every character on the line you entered. Did you accidentally capitalize `print` ? Did you forget one or both of the quotation marks or parentheses? Programming languages expect very specific syntax, and if you don't provide that, you'll get errors. If you can't get the program to run, see the suggestions in the next section.

Solución de problemas (Troubleshooting)

Si no consigues que *hello_world.py* se ejecute, aquí tienes algunos remedios que puedes probar y que también son buenas soluciones generales para cualquier problema de programación:

If you can't get *hello_world.py* to run, here are a few remedies you can try that are also good general solutions for any programming problem:

Listado en español:

- Cuando un programa contiene un error significativo, Python muestra un *traceback* , que es un informe de errores. Python examina el archivo e intenta identificar el problema. Comprueba el rastreo; puede darte una pista sobre qué problema impide que el programa se ejecute.
- Aléjate del ordenador, tómate un breve descanso y vuelve a intentarlo. Recuerda que la sintaxis es muy importante en programación, por lo que algo tan simple como unas comillas o unos paréntesis mal colocados puede impedir que un programa se ejecute correctamente. Vuelve a leer las partes relevantes de este capítulo, revisa tu código e intenta encontrar el error.
- Vuelve a empezar. Probablemente no necesites desinstalar ningún programa, pero podría tener sentido borrar tu archivo *hello_world.py* y volver a crearlo desde cero.
- Pide a otra persona que siga los pasos de este capítulo, en tu ordenador o en otro distinto, y observa atentamente lo que hace. Puede que te hayas saltado un pequeño paso que otra persona haya captado por casualidad.
- Consulta las instrucciones de instalación adicionales del Apéndice A ; algunos de los detalles incluidos en el Apéndice pueden ayudarte a resolver tu problema.

- Busca a alguien que sepa Python y pídele que te ayude a instalarlo. Si preguntas por ahí, puede que te encuentres con que conoces inesperadamente a alguien que utiliza Python.
- Las instrucciones de configuración de este capítulo también están disponibles a través del sitio web que acompaña a este libro en https://ehmatthes.github.io/pcc_3e . La versión online de estas instrucciones puede funcionar mejor porque puedes simplemente cortar y pegar código y hacer clic en los enlaces a los recursos que necesites.
- Pide ayuda en Internet. El Apéndice C proporciona una serie de recursos, como foros y sitios de chat en directo, donde puedes pedir soluciones a personas que ya han resuelto el problema al que te enfrentas actualmente.

Listing in English:

- When a program contains a significant error, Python displays a *traceback* , which is an error report. Python looks through the file and tries to identify the problem. Check the traceback; it might give you a clue as to what issue is preventing the program from running.
- Step away from your computer, take a short break, and then try again. Remember that syntax is very important in programming, so something as simple as mismatched quotation marks or mismatched parentheses can prevent a program from running properly. Reread the relevant parts of this chapter, look over your code, and try to find the mistake.
- Start over again. You probably don't need to uninstall any software, but it might make sense to delete your *hello_world.py* file and re-create it from scratch.
- Ask someone else to follow the steps in this chapter, on your computer or a different one, and watch what they do carefully. You might have missed one small step that someone else happens to catch.
- See the additional installation instructions in Appendix A ; some of the details included in the Appendix may help you solve your issue.
- Find someone who knows Python and ask them to help you get set up. If you ask around, you might find that you unexpectedly know someone who uses Python.
- The setup instructions in this chapter are also available through this book's companion website at https://ehmatthes.github.io/pcc_3e . The online version of these instructions might work better because you can simply cut and paste code and click links to the resources you need.
- Ask for help online. Appendix C provides a number of resources, such as forums and live chat sites, where you can ask for solutions from people who've already worked through the issue you're currently facing.

No te preocupes por molestar a programadores experimentados. Todos los programadores se han atascado alguna vez, y la mayoría de ellos estarán encantados de ayudarte a configurar correctamente tu sistema. Siempre que puedas exponer claramente lo que intentas hacer, lo que ya has probado y los resultados que obtienes, es muy probable que alguien pueda ayudarte. Como se ha mencionado en la introducción , la comunidad Python es muy amistosa y acogedora con los principiantes.

Never worry that you're bothering experienced programmers. Every programmer has been stuck at some point, and most programmers are happy to help you set up your system correctly. As long as you can state clearly what you're trying to do, what you've already tried, and the results you're getting, there's a good chance someone will be able to help you. As mentioned in the introduction, the Python community is very friendly and welcoming to beginners.

Python debería funcionar bien en cualquier ordenador moderno. Los primeros problemas de configuración pueden ser frustrantes, pero merece la pena resolverlos. Una vez que tengas *hello_world.py* funcionando, podrás empezar a aprender Python, y tu trabajo de programación será más interesante y satisfactorio.

Python should run well on any modern computer. Early setup issues can be frustrating, but they're well worth sorting out. Once you get *hello_world.py* running, you can start to learn Python, and your programming work will become more interesting and satisfying.

Ejecutar programas Python desde un terminal (*Running Python Programs from a Terminal*)

Ejecutarás la mayoría de tus programas directamente en tu editor de texto. Sin embargo, a veces es útil ejecutar programas desde un terminal. Por ejemplo, puedes querer ejecutar un programa existente sin abrirlo para editarlo.

You'll run most of your programs directly in your text editor. However, sometimes it's useful to run programs from a terminal instead. For example, you might want to run an existing program without opening it for editing.

Puedes hacerlo en cualquier sistema con Python instalado si sabes cómo acceder al directorio donde está almacenado el archivo del programa. Para intentarlo, asegúrate de que has guardado el archivo *hello_world.py* en la carpeta *python_work* de tu escritorio.

You can do this on any system with Python installed if you know how to access the directory where the program file is stored. To try this, make sure you've saved the *hello_world.py* file in the *python_work* folder on your desktop.

En Windows (On Windows)

Puedes utilizar el comando de terminal `cd` , para *change directory* , para navegar por tu sistema de archivos en una ventana de comandos. El comando `dir` , para *directory* , te muestra todos los archivos que existen en el directorio actual.

You can use the terminal command `cd` , for *change directory* , to navigate through your filesystem in a command window. The command `dir` , for *directory* , shows you all the files that exist in the current directory.

Abre una nueva ventana de terminal e introduce los siguientes comandos para ejecutar *hello_world.py* :

Open a new terminal window and enter the following commands to run *hello_world.py* :

```
C:\> cd Desktop\python_work
C:\Desktop\python_work> dir
hello_world.py
C:\Desktop\python_work> python hello_world.py
Hello Python world!
```

En primer lugar, utiliza el comando `cd` para navegar hasta la carpeta *python_work* , que se encuentra en la carpeta *Desktop* . A continuación, utiliza el comando `dir` para asegurarte de que *hello_world.py* se encuentra en esta carpeta. A continuación, ejecuta el archivo utilizando el comando `python hello_world.py` .

First, use the `cd` command to navigate to the *python_work* folder, which is in the *Desktop* folder. Next, use the `dir` command to make sure *hello_world.py* is in this folder. Then run the file using the command `python hello_world.py` .

La mayoría de tus programas funcionarán bien directamente desde el editor. Sin embargo, a medida que tu trabajo se vuelva más complejo, querrás ejecutar algunos de tus programas desde un terminal.

Most of your programs will run fine directly from your editor. However, as your work becomes more complex, you'll want to run some of your programs from a terminal.

En macOS y Linux (*On macOS and Linux*)

Ejecutar un programa Python desde una sesión de terminal es lo mismo en Linux que en macOS. Puedes utilizar el comando de terminal `cd` , para *change directory* , para navegar por tu sistema de archivos en una sesión de terminal. El comando `ls` , para *list* , te muestra todos los archivos no ocultos que existen en el directorio actual.

Running a Python program from a terminal session is the same on Linux and macOS. You can use the terminal command `cd` , for *change directory* , to navigate through your filesystem in a terminal session. The command `ls` , for *list* , shows you all the nonhidden files that exist in the current directory.

Abre una nueva ventana de terminal e introduce los siguientes comandos para ejecutar *hello_world.py* :

Open a new terminal window and enter the following commands to run *hello_world.py* :

```
~$ cd Desktop/python_work/  
~/Desktop/python_work$ ls  
hello_world.py  
~/Desktop/python_work$ python3 hello_world.py  
Hello Python world!
```

En primer lugar, utiliza el comando `cd` para navegar hasta la carpeta *python_work* , que se encuentra en la carpeta *Desktop* . A continuación, utiliza el comando `ls` para asegurarte de que *hello_world.py* está en esta carpeta. A continuación, ejecuta el archivo utilizando el comando `python3 hello_world.py` .

First, use the `cd` command to navigate to the *python_work* folder, which is in the *Desktop* folder. Next, use the `ls` command to make sure *hello_world.py* is in this folder. Then run the file using the command `python3 hello_world.py` .

La mayoría de tus programas funcionarán bien directamente desde tu editor. Pero a medida que tu trabajo se vuelva más complejo, querrás ejecutar algunos de tus programas desde un terminal.

Most of your programs will run fine directly from your editor. But as your work becomes more complex, you'll want to run some of your programs from a terminal.

Pruébalo tú mismo (*Try It Yourself*)

Los ejercicios de este capítulo son de naturaleza exploratoria. A partir del Capítulo 2, los retos que resolverás se basarán en lo que has aprendido.

The exercises in this chapter are exploratory in nature. Starting in Chapter 2, the challenges you'll solve will be based on what you've learned.

1-1. python.org: Explora la página principal de Python (<https://python.org>) para encontrar temas que te interesen. A medida que te familiarices con Python, las distintas partes del sitio te serán más útiles.

1-1. python.org: Explore the Python home page (<https://python.org>) to find topics that interest you. As you become familiar with Python, different parts of the site will be more useful to you.

1-2. Hola Mundo Typos: Abre el archivo *hello_world.py* que acabas de crear. Escribe una errata en alguna parte de la línea y vuelve a ejecutar el programa. ¿Puedes hacer una errata que genere un error? ¿Puedes entender el mensaje de error? ¿Puedes hacer una errata que no genere un error? ¿Por qué crees que no generó un error?

1-2. Hello World Typos: Open the *hello_world.py* file you just created. Make a typo somewhere in the line and run the program again. Can you make a typo that generates an error? Can you make sense of the error message? Can you make a typo that doesn't generate an error? Why do you think it didn't make an error?

1-3. Habilidades infinitas: Si tuvieras infinitas habilidades de programación, ¿qué construirías? Estás a punto de aprender a programar. Si tienes un objetivo final en mente, tendrás un uso inmediato para tus nuevas habilidades; ahora es un buen momento para escribir breves descripciones de lo que quieres crear. Es una buena costumbre llevar un cuaderno de "ideas" al que puedas acudir siempre que quieras empezar un nuevo proyecto. Dedica ahora unos minutos a describir tres programas que quieras crear.

1-3. Infinite Skills: If you had infinite programming skills, what would you build? You're about to learn how to program. If you have an end goal in mind, you'll have an immediate use for your new skills; now is a great time to write brief descriptions of what you want to create. It's a good habit to keep an "ideas" notebook that you can refer to whenever you want to start a new project. Take a few minutes now to describe three programs you want to create.

Resumen (Summary)

En este capítulo, has aprendido un poco sobre Python en general, y has instalado Python en tu sistema si aún no estaba allí. También has instalado un editor de texto para facilitar la escritura de código Python. Has ejecutado fragmentos de código Python en una sesión de terminal y has ejecutado tu primer programa, *hello_world.py*. Probablemente también aprendiste un poco sobre resolución de problemas.

In this chapter, you learned a bit about Python in general, and you installed Python on your system if it wasn't already there. You also installed a text editor to make it easier to write Python code. You ran snippets of Python code in a terminal session, and you ran your first program, *hello_world.py*. You probably learned a bit about troubleshooting as well.

En el próximo capítulo, aprenderás sobre los distintos tipos de datos con los que puedes trabajar en tus programas Python, y también empezarás a utilizar variables.

In the next chapter, you'll learn about the different kinds of data you can work with in your Python programs, and you'll start to use variables as well.

Capítulo 2: Variables y tipos de datos simples (Chapter 2: Variables and Simple Data Types)

2

Variables y tipos de datos simples (2 Variables and Simple Data Types)



DESCRIPCIÓN DE LA
IMAGEN (En español):

La imagen muestra una ilustración de una serpiente que está enroscada alrededor de la parte delantera de un motor a reacción. La serpiente tiene una expresión decidida y lleva gafas. No hay texto presente en la imagen.

En este capítulo aprenderás sobre los distintos tipos de datos con los que puedes trabajar en tus programas Python. También aprenderás a utilizar variables para representar datos en tus programas.

In this chapter you'll learn about the different kinds of data you can work with in your Python programs. You'll also learn how to use variables to represent data in your programs.

Qué ocurre realmente cuando ejecutas `hola_mundo.py` (What Really Happens When You Run `hello_world.py`)

Echemos un vistazo más de cerca a lo que hace Python cuando ejecutas `hello_world.py`. Resulta que Python hace bastante trabajo, incluso cuando ejecuta un programa sencillo:

Let's take a closer look at what Python does when you run `hello_world.py`. As it turns out, Python does a fair amount of work, even when it runs a simple program:

hola_mundo.py

hello_world.py

```
print("Hello Python world!")
```

Cuando ejecutes este código, deberías ver la siguiente salida:

When you run this code, you should see the following output:

Hello Python world!

Cuando ejecutas el archivo *hello_world.py*, la terminación *.py* indica que el archivo es un programa Python. A continuación, tu editor ejecuta el archivo a través del *Python interpreter*, que lee el programa y determina qué significa cada palabra del programa. Por ejemplo, cuando el intérprete ve la palabra `print` seguida de paréntesis, imprime en la pantalla lo que haya dentro de los paréntesis.

When you run the file *hello_world.py*, the ending *.py* indicates that the file is a Python program. Your editor then runs the file through the *Python interpreter*, which reads through the program and determines what each word in the program means. For example, when the interpreter sees the word `print` followed by parentheses, it prints to the screen whatever is inside the parentheses.

A medida que escribes tus programas, tu editor resalta diferentes partes de tu programa de diferentes maneras. Por ejemplo, reconoce que `print()` es el nombre de una función y muestra esa palabra en un color. Reconoce que `"Hello Python world!"` no es código Python y muestra esa frase en un color diferente. Esta función se llama *syntax highlighting* y es bastante útil cuando empiezas a escribir tus propios programas.

As you write your programs, your editor highlights different parts of your program in different ways. For example, it recognizes that `print()` is the name of a function and displays that word in one color. It recognizes that `"Hello Python world!"` is not Python code, and displays that phrase in a different color. This feature is called *syntax highlighting* and is quite useful as you start to write your own programs.

Variables (*Variables*)

Intentemos utilizar una variable en *hello_world.py* . Añade una nueva línea al principio del archivo, y modifica la segunda línea:

Let's try using a variable in *hello_world.py* . Add a new line at the beginning of the file, and modify the second line:

hola_mundo.py

hello_world.py

```
message = "Hello Python world!"  
print(message)
```

Ejecuta este programa para ver qué ocurre. Deberías ver la misma salida que viste anteriormente:

Run this program to see what happens. You should see the same output you saw previously:

Hello Python world!

Hemos añadido un *variable* llamado `message` . Cada variable está conectada a un *value* , que es la información asociada a esa variable. En este caso, el valor es el texto "Hello Python world!" .

We've added a *variable* named `message` . Every variable is connected to a *value* , which is the information associated with that variable. In this case the value is the "Hello Python world!" text.

Añadir una variable supone un poco más de trabajo para el intérprete de Python. Cuando procesa la primera línea, asocia la variable `message` al texto "Hello Python world!" . Cuando llega a la segunda línea, imprime en la pantalla el valor asociado a `message` .

Adding a variable makes a little more work for the Python interpreter. When it processes the first line, it associates the variable `message` with the "Hello Python world!" text. When it reaches the second line, it prints the value associated with `message` to the screen.

Vamos a ampliar este programa modificando *hello_world.py* para imprimir un segundo mensaje. Añade una línea en blanco a *hello_world.py* , y luego añade dos nuevas líneas de código:

Let's expand on this program by modifying *hello_world.py* to print a second message. Add a blank line to *hello_world.py* , and then add two new lines of code:

```
message = "Hello Python world!"  
print(message)
```

```
message = "Hello Python Crash Course world!"  
print(message)
```

Ahora, cuando ejecutes *hello_world.py* , deberías ver dos líneas de salida:

Now when you run *hello_world.py* , you should see two lines of output:

```
Hello Python world!  
Hello Python Crash Course world!
```

Puedes cambiar el valor de una variable en tu programa en cualquier momento, y Python siempre mantendrá un registro de su valor actual.

You can change the value of a variable in your program at any time, and Python will always keep track of its current value.

Nombrar y utilizar variables (*Naming and Using Variables*)

Cuando utilices variables en Python, debes respetar algunas reglas y directrices. Incumplir algunas de estas reglas provocará errores; otras directrices simplemente te ayudarán a escribir un código más fácil de leer y comprender. Asegúrate de tener en cuenta las siguientes reglas cuando trabajes con variables:

When you're using variables in Python, you need to adhere to a few rules and guidelines. Breaking some of these rules will cause errors; other guidelines just help you write code that's easier to read and understand. Be sure to keep the following rules in mind when working with variables:

Listado en español:

- Los nombres de las variables sólo pueden contener letras, números y guiones bajos. Pueden empezar por una letra o un guión bajo, pero no por un número. Por ejemplo, puedes llamar a una variable `message_1` pero no `1_message` .
- No se permiten espacios en los nombres de las variables, pero se pueden utilizar guiones bajos para separar palabras en los nombres de las variables. Por ejemplo, `greeting_message` funciona, pero `greeting message` provocará errores.
- Evita utilizar palabras clave de Python y nombres de funciones como nombres de variables. Por ejemplo, no utilices la palabra `print` como nombre de variable; Python la ha reservado para un fin programático concreto. (Consulta "Palabras clave de Python y funciones incorporadas" en la página 466 .)
- Los nombres de las variables deben ser cortos pero descriptivos. Por ejemplo, `name` es mejor que `n` , `student_name` es mejor que `s_n` , y `name_length` es mejor que `length_of_persons_name` .
- Ten cuidado al utilizar la letra minúscula *l* y la mayúscula *O* porque podrían confundirse con los números *1* y *0* .

Listing in English:

- Variable names can contain only letters, numbers, and underscores. They can start with a letter or an underscore, but not with a number. For instance, you can call a variable `message_1` but not `1_message` .
- Spaces are not allowed in variable names, but underscores can be used to separate words in variable names. For example, `greeting_message` works but `greeting message` will cause errors.
- Avoid using Python keywords and function names as variable names. For example, do not use the word `print` as a variable name; Python has reserved it for a particular programmatic purpose. (See "Python Keywords and Built-in Functions" on page 466 .)

- Variable names should be short but descriptive. For example, `name` is better than `n` , `student_name` is better than `s_n` , and `name_length` is better than `length_of_persons_name` .
- Be careful when using the lowercase letter `l` and the uppercase letter `O` because they could be confused with the numbers `1` and `0` .

Puede llevarte algo de práctica aprender a crear buenos nombres de variables, sobre todo a medida que tus programas se vuelvan más interesantes y complicados. A medida que escribas más programas y empieces a leer el código de otras personas, mejorarás en la creación de nombres significativos.

It can take some practice to learn how to create good variable names, especially as your programs become more interesting and complicated. As you write more programs and start to read through other people's code, you'll get better at coming up with meaningful names.

Nota (Note)

Las variables Python que utilices en este momento deben estar en minúsculas. No tendrás errores si utilizas mayúsculas, pero las mayúsculas en los nombres de variables tienen significados especiales que discutiremos en capítulos posteriores.

The Python variables you're using at this point should be lowercase. You won't get errors if you use uppercase letters, but uppercase letters in variable names have special meanings that we'll discuss in later chapters.

Evitar errores de nombre al utilizar variables (Avoiding Name Errors When Using Variables)

Todo programador comete errores, y la mayoría los comete todos los días. Aunque los buenos programadores pueden cometer errores, también saben responder a esos errores con eficacia. Veamos un error que es probable que cometas al principio y aprendamos a solucionarlo.

Every programmer makes mistakes, and most make mistakes every day. Although good programmers might create errors, they also know how to respond to those errors efficiently. Let's look at an error you're likely to make early on and learn how to fix it.

Escribiremos un código que genere un error a propósito. Introduce el siguiente código, incluyendo la palabra mal escrita `mesage` , que se muestra en negrita:

We'll write some code that generates an error on purpose. Enter the following code, including the misspelled word `mesage` , which is shown in bold:

```
message = "Hello Python Crash Course reader!"  
print(mesage)
```

Cuando se produce un error en tu programa, el intérprete de Python hace todo lo posible para ayudarte a averiguar dónde está el problema. El intérprete proporciona un rastreo cuando un programa no puede ejecutarse correctamente. Un *traceback* es un registro de dónde tuvo problemas el intérprete al intentar ejecutar tu código. He aquí un ejemplo del rastreo que proporciona Python después de que accidentalmente hayas escrito mal el nombre de una variable:

When an error occurs in your program, the Python interpreter does its best to help you figure out where the problem is. The interpreter provides a traceback when a program cannot run successfully. A *traceback* is a record of where the interpreter ran into trouble when trying to execute your code. Here's an example of the traceback that Python provides after you've accidentally misspelled a variable's name:

Traceback (most recent call last):

- ❶ File "hello_world.py", line 2, in <module>
- ❷ `print(mesage)`
 ^^^^^^
- ❸ `NameError: name 'mesage' is not defined. Did you mean: 'message'?`

La salida informa de que se produce un error en la línea 2 del archivo *hello_world.py* ❶ . El intérprete muestra esta línea ❷ para ayudarnos a detectar el error rápidamente y nos dice qué tipo de error ha encontrado ❸ . En este caso ha encontrado un *name error* e informa de que la variable que se está imprimiendo, `mesage` , no está definida. Python no puede identificar el nombre de la variable proporcionada. Un error de nombre suele significar que hemos olvidado definir el valor de una variable antes de utilizarla, o que

hemos cometido un error ortográfico al introducir el nombre de la variable. Si Python encuentra un nombre de variable similar al que no reconoce, te preguntará si ese es el nombre que querías utilizar.

The output reports that an error occurs in line 2 of the file *hello_world.py* ❶ . The interpreter shows this line ❷ to help us spot the error quickly and tells us what kind of error it found ❸ . In this case it found a *name error* and reports that the variable being printed, *mesage* , has not been defined. Python can't identify the variable name provided. A name error usually means we either forgot to set a variable's value before using it, or we made a spelling mistake when entering the variable's name. If Python finds a variable name that's similar to the one it doesn't recognize, it will ask if that's the name you meant to use.

En este ejemplo hemos omitido la letra *s* en el nombre de la variable *message* en la segunda línea. El intérprete de Python no revisa la ortografía de tu código, pero sí se asegura de que los nombres de las variables se escriban de forma coherente. Por ejemplo, observa lo que ocurre cuando escribimos *message* incorrectamente en la línea que define la variable:

In this example we omitted the letter *s* in the variable name *message* in the second line. The Python interpreter doesn't spellcheck your code, but it does ensure that variable names are spelled consistently. For example, watch what happens when we spell *message* incorrectly in the line that defines the variable:

```
mesage = "Hello Python Crash Course reader!"  
print(mesage)
```

En este caso, ¡el programa se ejecuta correctamente!

In this case, the program runs successfully!

Hello Python Crash Course reader!

Los nombres de las variables coinciden, así que Python no ve ningún problema. Los lenguajes de programación son estrictos, pero no tienen en cuenta la buena y la mala ortografía. En consecuencia, no necesitas tener en cuenta las reglas ortográficas y gramaticales del inglés cuando intentes crear nombres de variables y escribir código.

The variable names match, so Python sees no issue. Programming languages are strict, but they disregard good and bad spelling. As a result, you don't need to consider English spelling and grammar rules when you're trying to create variable names and writing code.

Muchos errores de programación son simples erratas de un solo carácter en una línea de un programa. Si te encuentras pasando mucho tiempo buscando uno de estos errores, debes saber que estás en buena compañía. Muchos programadores experimentados y con talento pasan horas buscando este tipo de errores minúsculos. Intenta reírte de ello y sigue adelante, sabiendo que ocurrirá con frecuencia a lo largo de tu vida de programador.

Many programming errors are simple, single-character typos in one line of a program. If you find yourself spending a long time searching for one of these errors, know that you're in good company. Many experienced and talented programmers spend hours hunting down these kinds of tiny errors. Try to laugh about it and move on, knowing it will happen frequently throughout your programming life.

Las variables son etiquetas (*Variables Are Labels*)

Las variables se describen a menudo como cajas en las que puedes almacenar valores. Esta idea puede ser útil las primeras veces que utilices una variable, pero no es una forma precisa de describir cómo se representan internamente las variables en Python. Es mucho mejor pensar en las variables como etiquetas que puedes asignar a valores. También puedes decir que una variable hace referencia a un determinado valor.

Variables are often described as boxes you can store values in. This idea can be helpful the first few times you use a variable, but it isn't an accurate way to describe how variables are represented internally in Python. It's much better to think of variables as labels that you can assign to values. You can also say that a variable references a certain value.

Esta distinción probablemente no tenga mucha importancia en tus programas iniciales, pero merece la pena aprenderla antes que después. En algún momento, verás un comportamiento inesperado de una variable, y una comprensión precisa de cómo funcionan las variables te ayudará a identificar lo que está ocurriendo en tu código.

This distinction probably won't matter much in your initial programs, but it's worth learning earlier rather than later. At some point, you'll see unexpected behavior from a variable, and an accurate understanding of how variables work will help you identify what's happening in your code.

Nota (Note)

La mejor forma de entender nuevos conceptos de programación es intentar utilizarlos en tus programas. Si te quedas atascado mientras trabajas en un ejercicio de este libro, intenta hacer otra cosa durante un rato. Si sigues atascado, repasa la parte correspondiente de ese capítulo. Si sigues necesitando ayuda, consulta las sugerencias del Apéndice C .

The best way to understand new programming concepts is to try using them in your programs. If you get stuck while working on an exercise in this book, try doing something else for a while. If you're still stuck, review the relevant part of that chapter. If you still need help, see the suggestions in Appendix C .

Inténtalo tú mismo (Try It Yourself)

Escribe un programa distinto para realizar cada uno de estos ejercicios. Guarda cada programa con un nombre de archivo que siga las convenciones estándar de Python, utilizando letras minúsculas y guiones bajos, como *simple_message.py* y *simple_messages.py* .

Write a separate program to accomplish each of these exercises. Save each program with a filename that follows standard Python conventions, using lowercase letters and underscores, such as *simple_message.py* and *simple_messages.py* .

2-1. Mensaje simple: Asigna un mensaje a una variable y luego imprime ese mensaje.

2-1. Simple Message: Assign a message to a variable, and then print that message.

2-2. Mensajes Simples: Asigna un mensaje a una variable, e imprime ese mensaje. A continuación, cambia el valor de la variable por un nuevo mensaje, e imprime el nuevo mensaje.

2-2. Simple Messages: Assign a message to a variable, and print that message. Then change the value of the variable to a new message, and print the new message.

Cadenas (Strings)

Dado que la mayoría de los programas definen y recogen algún tipo de dato y luego hacen algo útil con él, resulta útil clasificar los distintos tipos de datos. El primer tipo de datos que veremos es la cadena. Las cadenas son bastante sencillas a primera vista, pero puedes utilizarlas de muchas formas distintas.

Because most programs define and gather some sort of data and then do something useful with it, it helps to classify different types of data. The first data type we'll look at is the string. Strings are quite simple at first glance, but you can use them in many different ways.

Una *string* es una serie de caracteres. Cualquier cosa que esté entre comillas se considera una cadena en Python, y puedes utilizar comillas simples o dobles alrededor de tus cadenas, de esta forma:

A *string* is a series of characters. Anything inside quotes is considered a string in Python, and you can use single or double quotes around your strings like this:

```
"This is a string."  
'This is also a string.'
```

Esta flexibilidad te permite utilizar comillas y apóstrofes dentro de tus cadenas:

This flexibility allows you to use quotes and apostrophes within your strings:

'I told my friend, "Python is my favorite language!"
"The language 'Python' is named after Monty Python, not the snake."
"One of Python's strengths is its diverse and supportive community."

Exploremos algunas de las formas en que puedes utilizar cadenas.

Let's explore some of the ways you can use strings.

Cambiar mayúsculas y minúsculas en una cadena con métodos (*Changing Case in a String with Methods*)

Una de las tareas más sencillas que puedes hacer con las cadenas es cambiar las mayúsculas y minúsculas de las palabras de una cadena. Mira el siguiente código e intenta determinar qué está ocurriendo:

One of the simplest tasks you can do with strings is change the case of the words in a string. Look at the following code, and try to determine what's happening:

nombre.py

name.py

```
name = "ada lovelace"  
print(name.title())
```

Guarda este archivo como *name.py* y ejecútalo. Deberías ver esta salida:

Save this file as *name.py* and then run it. You should see this output:

Ada Lovelace

En este ejemplo, la variable `name` hace referencia a la cadena en minúsculas `"ada lovelace"`. El método `title()` aparece después de la variable en la llamada `print()`. Un

method es una acción que Python puede realizar sobre un dato. El punto (.) después de `name` en `name.title()` indica a Python que haga que el método `title()` actúe sobre la variable `name` . Cada método va seguido de un conjunto de paréntesis, porque los métodos a menudo necesitan información adicional para realizar su trabajo. Esa información se proporciona dentro de los paréntesis. La función `title()` no necesita ninguna información adicional, por lo que sus paréntesis están vacíos.

In this example, the variable `name` refers to the lowercase string "ada lovelace" . The method `title()` appears after the variable in the `print()` call. A *method* is an action that Python can perform on a piece of data. The dot (.) after `name` in `name.title()` tells Python to make the `title()` method act on the variable `name` . Every method is followed by a set of parentheses, because methods often need additional information to do their work. That information is provided inside the parentheses. The `title()` function doesn't need any additional information, so its parentheses are empty.

El método `title()` cambia cada palabra a mayúsculas, donde cada palabra empieza con mayúscula. Esto es útil porque a menudo querrás pensar en un nombre como una información. Por ejemplo, puede que quieras que tu programa reconozca los valores de entrada `Ada` , `ADA` , y `ada` como el mismo nombre, y los muestre todos como `Ada` .

The `title()` method changes each word to title case, where each word begins with a capital letter. This is useful because you'll often want to think of a name as a piece of information. For example, you might want your program to recognize the input values `Ada` , `ADA` , and `ada` as the same name, and display all of them as `Ada` .

También hay otros métodos útiles para tratar las mayúsculas y minúsculas. Por ejemplo, puedes cambiar una cadena a todo mayúsculas o todo minúsculas de esta forma:

Several other useful methods are available for dealing with case as well. For example, you can change a string to all uppercase or all lowercase letters like this:

```
name = "Ada Lovelace"
print(name.upper())
print(name.lower())
```

Esto mostrará lo siguiente:

This will display the following:

```
ADA LOVELACE  
ada lovelace
```

El método `lower()` es especialmente útil para almacenar datos. Normalmente no querrás fiarte de las mayúsculas que te proporcionen tus usuarios, así que convertirás las cadenas a minúsculas antes de almacenarlas. Luego, cuando quieras mostrar la información, utilizarás las mayúsculas y minúsculas que tengan más sentido para cada cadena.

The `lower()` method is particularly useful for storing data. You typically won't want to trust the capitalization that your users provide, so you'll convert strings to lowercase before storing them. Then when you want to display the information, you'll use the case that makes the most sense for each string.

Uso de variables en cadenas *(Using Variables in Strings)*

En algunas situaciones, querrás utilizar el valor de una variable dentro de una cadena. Por ejemplo, podrías querer utilizar dos variables para representar un nombre y un apellido, respectivamente, y luego combinar esos valores para mostrar el nombre completo de alguien:

In some situations, you'll want to use a variable's value inside a string. For example, you might want to use two variables to represent a first name and a last name, respectively, and then combine those values to display someone's full name:

nombre_completo.py

full_name.py

```
first_name = "ada"  
last_name = "lovelace"
```



```
❶ full_name = f'{first_name} {last_name}'  
print(full_name)
```

Para insertar el valor de una variable en una cadena, coloca la letra `f` inmediatamente antes de la comilla de apertura ❶ . Pon llaves alrededor del nombre o nombres de cualquier variable que quieras utilizar dentro de la cadena. Python sustituirá cada variable por su valor cuando se muestre la cadena.

To insert a variable's value into a string, place the letter `f` immediately before the opening quotation mark ❶ . Put braces around the name or names of any variable you want to use inside the string. Python will replace each variable with its value when the string is displayed.

Estas cadenas se llaman *f-strings* . El *f* es para *format* , porque Python formatea la cadena sustituyendo el nombre de cualquier variable entre llaves por su valor. La salida del código anterior es

These strings are called *f-strings* . The *f* is for *format* , because Python formats the string by replacing the name of any variable in braces with its value. The output from the previous code is:

```
ada lovelace
```

Puedes hacer muchas cosas con las cadenas `f`. Por ejemplo, puedes utilizar cadenas `f` para componer mensajes completos utilizando la información asociada a una variable, como se muestra aquí:

You can do a lot with `f-strings`. For example, you can use `f-strings` to compose complete messages using the information associated with a variable, as shown here:

```
first_name = "ada"  
last_name = "lovelace"  
full_name = f'{first_name} {last_name}'  
❶ print(f'Hello, {full_name.title()}!')
```

El nombre completo se utiliza en una frase que saluda al usuario ❶ , y el método `title()` cambia el nombre a mayúsculas y minúsculas. Este código devuelve un saludo sencillo pero

bien formateado:

The full name is used in a sentence that greets the user ❶ , and the `title()` method changes the name to title case. This code returns a simple but nicely formatted greeting:

Hello, Ada Lovelace!

También puedes utilizar cadenas `f` para componer un mensaje, y luego asignar todo el mensaje a una variable:

You can also use f-strings to compose a message, and then assign the entire message to a variable:

```
first_name = "ada"
last_name = "lovelace"
full_name = f'{first_name} {last_name}'
❶ message = f'Hello, {full_name.title()}!'
❷ print(message)
```

Este código también muestra el mensaje `Hello, Ada Lovelace!` , pero al asignar el mensaje a una variable ❶ hacemos que la llamada final a `print()` sea mucho más sencilla ❷ .

This code displays the message `Hello, Ada Lovelace!` as well, but by assigning the message to a variable ❶ we make the final `print()` call much simpler ❷ .

Añadir espacios en blanco a las cadenas con tabuladores o nuevas líneas *(Adding Whitespace to Strings with Tabs or Newlines)*

En programación, *whitespace* se refiere a cualquier carácter no impreso, como espacios, tabuladores y símbolos de fin de línea. Puedes utilizar los espacios en blanco para organizar tu salida de forma que sea más fácil de leer para los usuarios.

In programming, *whitespace* refers to any nonprinting characters, such as spaces, tabs, and end-of-line symbols. You can use whitespace to organize your output so it's easier for users to read.

Para añadir un tabulador a tu texto, utiliza la combinación de caracteres `\t` :

To add a tab to your text, use the character combination `\t` :

```
>>> print("Python")
Python
>>> print("\tPython")
    Python
```

Para añadir una nueva línea en una cadena, utiliza la combinación de caracteres `\n` :

To add a newline in a string, use the character combination `\n` :

```
>>> print("Languages:\nPython\nC\nJavaScript")
Languages:
Python
C
JavaScript
```

También puedes combinar tabuladores y nuevas líneas en una sola cadena. La cadena `"\n\t"` le dice a Python que se mueva a una nueva línea y comience la siguiente con un tabulador. El siguiente ejemplo muestra cómo puedes utilizar una cadena de una sola línea para generar cuatro líneas de salida:

You can also combine tabs and newlines in a single string. The string `"\n\t"` tells Python to move to a new line, and start the next line with a tab. The following example shows how you can use a one-line string to generate four lines of output:

```
>>> print("Languages:\n\tPython\n\tC\n\tJavaScript")
Languages:
    Python
    C
    JavaScript
```

Las nuevas líneas y los tabuladores serán muy útiles en los dos próximos capítulos, cuando empieces a producir muchas líneas de salida a partir de unas pocas líneas de código.

Newlines and tabs will be very useful in the next two chapters, when you start to produce many lines of output from just a few lines of code.

Eliminar espacios en blanco (*Stripping Whitespace*)

Los espacios en blanco adicionales pueden confundir en tus programas. Para los programadores, `'python'` y `'python '` son prácticamente lo mismo. Pero para un programa, son dos cadenas diferentes. Python detecta el espacio extra en `'python '` y lo considera significativo a menos que le digas lo contrario.

Extra whitespace can be confusing in your programs. To programmers, `'python'` and `'python '` look pretty much the same. But to a program, they are two different strings. Python detects the extra space in `'python '` and considers it significant unless you tell it otherwise.

Es importante pensar en los espacios en blanco, porque a menudo querrás comparar dos cadenas para determinar si son iguales. Por ejemplo, un caso importante podría consistir en comprobar los nombres de usuario de las personas cuando se conectan a un sitio web. Los espacios en blanco adicionales también pueden confundir en situaciones mucho más sencillas. Afortunadamente, Python facilita la eliminación de los espacios en blanco sobrantes de los datos que introduce la gente.

It's important to think about whitespace, because often you'll want to compare two strings to determine whether they are the same. For example, one important instance might involve checking people's usernames when they log in to a website. Extra whitespace can be confusing in much simpler situations as well. Fortunately, Python makes it easy to eliminate extra whitespace from data that people enter.

Python puede buscar espacios en blanco adicionales en los lados derecho e izquierdo de una cadena. Para asegurarte de que no hay espacios en blanco en el lado derecho de una cadena, utiliza el método `rstrip()` :

Python can look for extra whitespace on the right and left sides of a string. To ensure that no whitespace exists at the right side of a string, use the `rstrip()` method:

```
❶ >>> favorite_language = 'python '  
❷ >>> favorite_language  
'python '  
❸ >>> favorite_language.rstrip()  
'python'  
❹ >>> favorite_language  
'python '
```

El valor asociado a `favorite_language` ❶ contiene espacios en blanco adicionales al final de la cadena. Cuando pides a Python este valor en una sesión de terminal, puedes ver el espacio al final del valor ❷. Cuando el método `rstrip()` actúa sobre la variable `favorite_language` ❸, este espacio extra se elimina. Sin embargo, sólo se elimina temporalmente. Si vuelves a preguntar por el valor de `favorite_language`, la cadena tendrá el mismo aspecto que cuando se introdujo, incluido el espacio en blanco extra ❹.

The value associated with `favorite_language` ❶ contains extra whitespace at the end of the string. When you ask Python for this value in a terminal session, you can see the space at the end of the value ❷. When the `rstrip()` method acts on the variable `favorite_language` ❸, this extra space is removed. However, it is only removed temporarily. If you ask for the value of `favorite_language` again, the string looks the same as when it was entered, including the extra whitespace ❹.

Para eliminar los espacios en blanco de la cadena de forma permanente, tienes que asociar el valor eliminado al nombre de la variable:

To remove the whitespace from the string permanently, you have to associate the stripped value with the variable name:

```
>>> favorite_language = 'python '  
❶ >>> favorite_language = favorite_language.rstrip()  
>>> favorite_language  
'python'
```

Para eliminar los espacios en blanco de la cadena, elimina los espacios en blanco de la parte derecha de la cadena y asocia este nuevo valor a la variable original ❶. Cambiar el valor

de una variable se hace a menudo en programación. Así es como se puede actualizar el valor de una variable mientras se ejecuta un programa o en respuesta a una entrada del usuario.

To remove the whitespace from the string, you strip the whitespace from the right side of the string and then associate this new value with the original variable ❶ . Changing a variable's value is done often in programming. This is how a variable's value can be updated as a program is executed or in response to user input.

También puedes eliminar los espacios en blanco del lado izquierdo de una cadena utilizando el método `rstrip()` , o de ambos lados a la vez utilizando `strip()` :

You can also strip whitespace from the left side of a string using the `rstrip()` method, or from both sides at once using `strip()` :

```
❶ >>> favorite_language = ' python '  
❷ >>> favorite_language.rstrip()  
' python'  
❸ >>> favorite_language.lstrip()  
'python '  
❹ >>> favorite_language.strip()  
'python'
```

En este ejemplo, empezamos con un valor que tiene espacios en blanco al principio y al final ❶ . A continuación, eliminamos el espacio sobrante del lado derecho ❷ , del lado izquierdo ❸ , y de ambos lados ❹ . Experimentar con estas funciones de eliminación puede ayudarte a familiarizarte con la manipulación de cadenas. En el mundo real, estas funciones de eliminación se utilizan sobre todo para limpiar las entradas del usuario antes de almacenarlas en un programa.

In this example, we start with a value that has whitespace at the beginning and the end ❶ . We then remove the extra space from the right side ❷ , from the left side ❸ , and from both sides ❹ . Experimenting with these stripping functions can help you become familiar with manipulating strings. In the real world, these stripping functions are used most often to clean up user input before it's stored in a program.

Eliminar prefijos (Removing Prefixes)

Al trabajar con cadenas, otra tarea común es eliminar un prefijo. Considera una URL con el prefijo común *https://* . Queremos eliminar este prefijo para centrarnos sólo en la parte de la URL que los usuarios deben introducir en la barra de direcciones. He aquí cómo hacerlo:

When working with strings, another common task is to remove a prefix. Consider a URL with the common prefix *https://* . We want to remove this prefix, so we can focus on just the part of the URL that users need to enter into an address bar. Here's how to do that:

```
>>> nostarch_url = 'https://nostarch.com'
>>> nostarch_url.removeprefix('https://')
'nostarch.com'
```

Introduce el nombre de la variable seguido de un punto y, a continuación, el método `removeprefix()` . Dentro del paréntesis, introduce el prefijo que quieras eliminar de la cadena original.

Enter the name of the variable followed by a dot, and then the method `removeprefix()` . Inside the parentheses, enter the prefix you want to remove from the original string.

Al igual que los métodos para eliminar espacios en blanco, `removeprefix()` no modifica la cadena original. Si quieres conservar el nuevo valor con el prefijo eliminado, reasígnalo a la variable original o asígnalo a una nueva variable:

Like the methods for removing whitespace, `removeprefix()` leaves the original string unchanged. If you want to keep the new value with the prefix removed, either reassign it to the original variable or assign it to a new variable:

```
>>> simple_url = nostarch_url.removeprefix('https://')
```

Cuando veas una URL en una barra de direcciones y no aparezca la parte *https://* , probablemente el navegador esté utilizando un método como `removeprefix()` entre bastidores.

When you see a URL in an address bar and the *https://* part isn't shown, the browser is probably using a method like `removeprefix()` behind the scenes.

Evitar errores de sintaxis con cadenas (Avoiding Syntax Errors with Strings)

Un tipo de error que puedes ver con cierta regularidad es un error de sintaxis. Un *syntax error* se produce cuando Python no reconoce una sección de tu programa como código Python válido. Por ejemplo, si utilizas un apóstrofo entre comillas simples, se producirá un error. Esto ocurre porque Python interpreta todo lo que hay entre la primera comilla simple y el apóstrofo como una cadena. Luego intenta interpretar el resto del texto como código Python, lo que provoca errores.

One kind of error that you might see with some regularity is a syntax error. A *syntax error* occurs when Python doesn't recognize a section of your program as valid Python code. For example, if you use an apostrophe within single quotes, you'll produce an error. This happens because Python interprets everything between the first single quote and the apostrophe as a string. It then tries to interpret the rest of the text as Python code, which causes errors.

A continuación te explicamos cómo utilizar correctamente las comillas simples y dobles. Guarda este programa como *apostrophe.py* y ejecútalo:

Here's how to use single and double quotes correctly. Save this program as *apostrophe.py* and then run it:

apostrofe.py

apostrophe.py

```
message = "One of Python's strengths is its diverse community."  
print(message)
```

El apóstrofo aparece dentro de un conjunto de comillas dobles, por lo que el intérprete de Python no tiene problemas para leer correctamente la cadena:

The apostrophe appears inside a set of double quotes, so the Python interpreter has no trouble reading the string correctly:

One of Python's strengths is its diverse community.

Sin embargo, si utilizas comillas simples, Python no puede identificar dónde debe terminar la cadena:

However, if you use single quotes, Python can't identify where the string should end:

```
message = 'One of Python's strengths is its diverse community.'  
print(message)
```

Verás la siguiente salida:

You'll see the following output:

```
File "apostrophe.py", line 1  
  message = 'One of Python's strengths is its diverse community.'  
                                     ❶ ^
```

SyntaxError: unterminated string literal (detected at line 1)

En la salida puedes ver que el error se produce justo después de la comilla simple final ❶ . Este error de sintaxis indica que el intérprete no reconoce algo en el código como código Python válido, y piensa que el problema puede ser una cadena que no está entrecomillada correctamente. Los errores pueden provenir de diversas fuentes, y señalaré algunas comunes a medida que surjan. Es posible que veas errores de sintaxis con frecuencia a medida que aprendas a escribir código Python correctamente. Los errores de sintaxis son también el tipo de error menos específico, por lo que pueden ser difíciles y frustrantes de identificar y corregir. Si te quedas atascado en un error especialmente persistente, consulta las sugerencias del Apéndice C .

In the output you can see that the error occurs right after the final single quote ❶ . This syntax error indicates that the interpreter doesn't recognize something in the code as valid Python code, and it thinks the problem might be a string that's not quoted correctly. Errors can come from a variety of sources, and I'll point out some common ones as they arise. You might see syntax errors often as you learn to write proper Python code. Syntax

errors are also the least specific kind of error, so they can be difficult and frustrating to identify and correct. If you get stuck on a particularly stubborn error, see the suggestions in Appendix C .

Nota (Note)

La función de resaltado de sintaxis de tu editor debería ayudarte a detectar rápidamente algunos errores de sintaxis mientras escribes tus programas. Si ves código Python resaltado como si fuera inglés o inglés resaltado como si fuera código Python, probablemente tengas una comilla mal colocada en algún lugar de tu archivo.

Your editor's syntax highlighting feature should help you spot some syntax errors quickly as you write your programs. If you see Python code highlighted as if it's English or English highlighted as if it's Python code, you probably have a mismatched quotation mark somewhere in your file.

Pruébalo tú mismo (Try It Yourself)

Guarda cada uno de los siguientes ejercicios como un archivo independiente, con un nombre como *name_cases.py* . Si te atascas, tómate un descanso o consulta las sugerencias del Apéndice C .

Save each of the following exercises as a separate file, with a name like *name_cases.py* . If you get stuck, take a break or see the suggestions in Appendix C .

2-3. Mensaje personal: Utiliza una variable para representar el nombre de una persona, e imprime un mensaje para esa persona. Tu mensaje debe ser sencillo, como "Hola Eric, ¿te gustaría aprender algo de Python hoy?".

2-3. Personal Message: Use a variable to represent a person's name, and print a message to that person. Your message should be simple, such as, "Hello Eric, would you like to learn some Python today?"

2-4. Casos de nombre: Utiliza una variable para representar el nombre de una persona, y luego imprime el nombre de esa persona en minúsculas, mayúsculas y mayúsculas de título.

2-4. Name Cases: Use a variable to represent a person's name, and then print that person's name in lowercase, uppercase, and title case.

2-5. Cita célebre: Busca una cita de una persona famosa a la que admires. Imprime la cita y el nombre de su autor. Tu resultado debe ser algo parecido a lo siguiente, incluidas las comillas:

2-5. Famous Quote: Find a quote from a famous person you admire. Print the quote and the name of its author. Your output should look something like the following, including the quotation marks:

Albert Einstein dijo una vez: "Una persona que nunca cometió un error nunca intentó nada nuevo".

Albert Einstein once said, "A person who never made a mistake never tried anything new."

2-6. Cita célebre 2: Repite el Ejercicio 2-5, pero esta vez, representa el nombre de la persona célebre utilizando una variable llamada `famous_person` . Luego redacta tu mensaje y represéntalo con una nueva variable llamada `message` . Imprime tu mensaje.

2-6. Famous Quote 2: Repeat Exercise 2-5, but this time, represent the famous person's name using a variable called `famous_person` . Then compose your message and represent it with a new variable called `message` . Print your message.

2-7. Despojar nombres: Utiliza una variable para representar el nombre de una persona e incluye algunos caracteres de espacio en blanco al principio y al final del nombre. Asegúrate de utilizar cada combinación de caracteres, `"t"` y `"n"` , al menos una vez.

2-7. Stripping Names: Use a variable to represent a person's name, and include some whitespace characters at the beginning and end of the name. Make sure you use each character combination, `"\t"` and `"\n"` , at least once.

Imprime el nombre una vez, para que se muestren los espacios en blanco alrededor del nombre. A continuación, imprime el nombre utilizando cada una de las tres funciones de eliminación, `lstrip()` , `rstrip()` , y `strip()` .

Print the name once, so the whitespace around the name is displayed. Then print the name using each of the three stripping functions, `lstrip()` , `rstrip()` , and `strip()` .

2-8. Extensiones de archivo: Python tiene un método `removesuffix()` que funciona exactamente igual que `removeprefix()` . Asigna el valor `'python_notes.txt'` a una variable llamada `filename` . Luego utiliza el método `removesuffix()` para mostrar el nombre del archivo sin la extensión, como hacen algunos exploradores de archivos.

2-8. File Extensions: Python has a `removesuffix()` method that works exactly like `removeprefix()` . Assign the value `'python_notes.txt'` to a variable called `filename` . Then use the `removesuffix()` method to display the filename without the file extension, like some file browsers do.

Números (*Numbers*)

Los números se utilizan muy a menudo en programación para llevar la puntuación en los juegos, representar datos en visualizaciones, almacenar información en aplicaciones web, etc. Python trata los números de varias formas distintas, dependiendo de cómo se utilicen. Veamos primero cómo gestiona Python los números enteros, porque son los más sencillos de trabajar.

Numbers are used quite often in programming to keep score in games, represent data in visualizations, store information in web applications, and so on. Python treats numbers in

several different ways, depending on how they're being used. Let's first look at how Python manages integers, because they're the simplest to work with.

Números enteros (*Integers*)

En Python puedes sumar (+), restar (-), multiplicar (*) y dividir (/) enteros.

You can add (+), subtract (-), multiply (*), and divide (/) integers in Python.

```
>>> 2 + 3
5
>>> 3 - 2
1
>>> 2 * 3
6
>>> 3 / 2
1.5
```

En una sesión terminal, Python simplemente devuelve el resultado de la operación. Python utiliza dos símbolos de multiplicación para representar exponentes:

In a terminal session, Python simply returns the result of the operation. Python uses two multiplication symbols to represent exponents:

```
>>> 3 ** 2
9
>>> 3 ** 3
27
>>> 10 ** 6
1000000
```

Python también admite el orden de las operaciones, por lo que puedes utilizar varias operaciones en una misma expresión. También puedes utilizar paréntesis para modificar el orden de las operaciones, de modo que Python pueda evaluar tu expresión en el orden que especifiques. Por ejemplo:

Python supports the order of operations too, so you can use multiple operations in one expression. You can also use parentheses to modify the order of operations so Python can evaluate your expression in the order you specify. For example:

```
>>> 2 + 3*4
14
>>> (2 + 3) * 4
20
```

El espaciado en estos ejemplos no afecta al modo en que Python evalúa las expresiones; simplemente te ayuda a detectar más rápidamente las operaciones que tienen prioridad cuando lees el código.

The spacing in these examples has no effect on how Python evaluates the expressions; it simply helps you more quickly spot the operations that have priority when you're reading through the code.

Flotantes (*Floats*)

Python llama a cualquier número con punto decimal *float* . Este término se utiliza en la mayoría de los lenguajes de programación, y se refiere al hecho de que un punto decimal puede aparecer en cualquier posición de un número. Todo lenguaje de programación debe estar cuidadosamente diseñado para gestionar correctamente los números decimales, de modo que los números se comporten adecuadamente, independientemente de dónde aparezca el punto decimal.

Python calls any number with a decimal point a *float* . This term is used in most programming languages, and it refers to the fact that a decimal point can appear at any position in a number. Every programming language must be carefully designed to properly manage decimal numbers so numbers behave appropriately, no matter where the decimal point appears.

En su mayor parte, puedes utilizar números flotantes sin preocuparte de cómo se comportan. Simplemente introduce los números que quieras utilizar, y lo más probable es que Python haga lo que esperas:

For the most part, you can use floats without worrying about how they behave. Simply enter the numbers you want to use, and Python will most likely do what you expect:

```
>>> 0.1 + 0.1
0.2
>>> 0.2 + 0.2
0.4
>>> 2 * 0.1
0.2
>>> 2 * 0.2
0.4
```

Sin embargo, ten en cuenta que a veces puedes obtener un número arbitrario de decimales en tu respuesta:

However, be aware that you can sometimes get an arbitrary number of decimal places in your answer:

```
>>> 0.2 + 0.1
0.30000000000000004
>>> 3 * 0.1
0.30000000000000004
```

Esto ocurre en todos los lenguajes y no tiene mayor importancia. Python intenta encontrar una forma de representar el resultado con la mayor precisión posible, lo que a veces es difícil dado cómo los ordenadores tienen que representar los números internamente. Ignora los decimales de más por ahora; aprenderás formas de tratar con los decimales de más cuando lo necesites en los proyectos de la Parte II .

This happens in all languages and is of little concern. Python tries to find a way to represent the result as precisely as possible, which is sometimes difficult given how computers have to represent numbers internally. Just ignore the extra decimal places for now; you'll learn ways to deal with the extra places when you need to in the projects in Part II .

Enteros y flotantes (*Integers and Floats*)

Cuando divides dos números cualesquiera, aunque sean enteros y den como resultado un número entero, siempre obtendrás un flotante:

When you divide any two numbers, even if they are integers that result in a whole number, you'll always get a float:

```
>>> 4/2
2.0
```

Si mezclas un entero y un flotante en cualquier otra operación, obtendrás también un flotante:

If you mix an integer and a float in any other operation, you'll get a float as well:

```
>>> 1 + 2.0
3.0
>>> 2 * 3.0
6.0
>>> 3.0 ** 2
9.0
```

Python utiliza por defecto un flotador en cualquier operación que utilice un flotador, aunque el resultado sea un número entero.

Python defaults to a float in any operation that uses a float, even if the output is a whole number.

Guiones bajos en los números (*Underscores in Numbers*)

Cuando escribas números largos, puedes agrupar los dígitos utilizando guiones bajos para que los números grandes sean más legibles:

When you're writing long numbers, you can group digits using underscores to make large numbers more readable:

```
>>> universe_age = 14_000_000_000
```


Cuando imprimes un número definido con guiones bajos, Python sólo imprime los dígitos:

When you print a number that was defined using underscores, Python prints only the digits:

```
>>> print(universe_age)
14000000000
```

Python ignora los guiones bajos cuando almacena este tipo de valores. Aunque no agrupes los dígitos de tres en tres, el valor no se verá afectado. Para Python, 1000 es lo mismo que 1_000 , que es lo mismo que 10_00 . Esta característica funciona tanto para enteros como para flotantes.

Python ignores the underscores when storing these kinds of values. Even if you don't group the digits in threes, the value will still be unaffected. To Python, 1000 is the same as 1_000 , which is the same as 10_00 . This feature works for both integers and floats.

Asignación múltiple *(Multiple Assignment)*

Puedes asignar valores a más de una variable utilizando una sola línea de código. Esto puede ayudar a acortar tus programas y hacerlos más fáciles de leer; utilizarás esta técnica con más frecuencia al inicializar un conjunto de números.

You can assign values to more than one variable using just a single line of code. This can help shorten your programs and make them easier to read; you'll use this technique most often when initializing a set of numbers.

Por ejemplo, así es como puedes inicializar a cero las variables `x` , `y` , y `z` :

For example, here's how you can initialize the variables `x` , `y` , and `z` to zero:

```
>>> x, y, z = 0, 0, 0
```

Tienes que separar los nombres de las variables con comas, y hacer lo mismo con los valores, y Python asignará cada valor a su respectiva variable. Siempre que el número de valores coincida con el número de variables, Python las emparejará correctamente.

You need to separate the variable names with commas, and do the same with the values, and Python will assign each value to its respective variable. As long as the number of values matches the number of variables, Python will match them up correctly.

Constantes (*Constants*)

Una *constant* es una variable cuyo valor permanece invariable a lo largo de la vida de un programa. Python no tiene tipos de constantes incorporados, pero los programadores de Python utilizan letras mayúsculas para indicar que una variable debe tratarse como una constante y no modificarse nunca:

A *constant* is a variable whose value stays the same throughout the life of a program. Python doesn't have built-in constant types, but Python programmers use all capital letters to indicate a variable should be treated as a constant and never be changed:

```
MAX_CONNECTIONS = 5000
```

Cuando quieras tratar una variable como una constante en tu código, escribe el nombre de la variable en mayúsculas.

When you want to treat a variable as a constant in your code, write the name of the variable in all capital letters.

Pruébalo tú mismo (*Try It Yourself*)

2-9. Número Ocho: Escribe operaciones de suma, resta, multiplicación y división que den como resultado el número 8. Asegúrate de encerrar tus operaciones en llamadas a `print()` para ver los resultados. Debes crear cuatro líneas con este aspecto:

2-9. Number Eight: Write addition, subtraction, multiplication, and division operations that each result in the number 8. Be sure to enclose your operations in `print()` calls to see the results. You should create four lines that look like this:

```
print(5+3)
```

Tus resultados deben ser cuatro líneas, con el número 8 apareciendo una vez en cada línea.

Your output should be four lines, with the number 8 appearing once on each line.

2-10. Número favorito: Utiliza una variable para representar tu número favorito. Luego, utilizando esa variable, crea un mensaje que revele tu número favorito. Imprime ese mensaje.

2-10. Favorite Number: Use a variable to represent your favorite number. Then, using that variable, create a message that reveals your favorite number. Print that message.

Comentarios (*Comments*)

Los comentarios son una característica extremadamente útil en la mayoría de los lenguajes de programación. Todo lo que has escrito en tus programas hasta ahora es código Python. A medida que tus programas se hacen más largos y complicados, debes añadir notas dentro de tus programas que describan tu enfoque general del problema que estás resolviendo. Un *comment* te permite escribir notas en tu lenguaje hablado, dentro de tus programas.

Comments are an extremely useful feature in most programming languages. Everything you've written in your programs so far is Python code. As your programs become longer and more complicated, you should add notes within your programs that describe your overall approach to the problem you're solving. A *comment* allows you to write notes in your spoken language, within your programs.

¿Cómo se escriben los comentarios? *(How Do You Write Comments?)*

En Python, la almohadilla (#) indica un comentario. Cualquier cosa que siga a una almohadilla en tu código es ignorada por el intérprete de Python. Por ejemplo

In Python, the hash mark (#) indicates a comment. Anything following a hash mark in your code is ignored by the Python interpreter. For example:

comentario.py

comment.py

```
# Say hello to everyone.  
print("Hello Python people!")
```

Python ignora la primera línea y ejecuta la segunda.

Python ignores the first line and executes the second line.

Hello Python people!

¿Qué tipo de comentarios debes escribir? *(What Kinds of Comments Should You Write?)*

La razón principal para escribir comentarios es explicar qué se supone que debe hacer tu código y cómo lo estás haciendo funcionar. Cuando estás trabajando en un proyecto, entiendes cómo encajan todas las piezas. Pero cuando vuelves al proyecto después de un tiempo de ausencia, es probable que hayas olvidado algunos detalles en . Siempre puedes estudiar tu código durante un tiempo y averiguar cómo se suponía que debían funcionar los segmentos, pero escribir buenos comentarios puede ahorrarte tiempo al resumir claramente tu planteamiento general.

The main reason to write comments is to explain what your code is supposed to do and how you are making it work. When you're in the middle of working on a project, you understand how all of the pieces fit together. But when you return to a project after some time away, you'll likely have forgotten some of the details. You can always study your code

for a while and figure out how segments were supposed to work, but writing good comments can save you time by summarizing your overall approach clearly.

Si quieres convertirte en un programador profesional o colaborar con otros programadores, debes escribir comentarios significativos. Hoy en día, la mayoría del software se escribe en colaboración, ya sea por un grupo de empleados de una empresa o por un grupo de personas que trabajan juntas en un proyecto de código abierto. Los programadores cualificados esperan ver comentarios en el código, así que es mejor que empieces a añadir comentarios descriptivos a tus programas ahora. Escribir comentarios claros y concisos en tu código es uno de los hábitos más beneficiosos que puedes adquirir como nuevo programador.

If you want to become a professional programmer or collaborate with other programmers, you should write meaningful comments. Today, most software is written collaboratively, whether by a group of employees at one company or a group of people working together on an open source project. Skilled programmers expect to see comments in code, so it's best to start adding descriptive comments to your programs now. Writing clear, concise comments in your code is one of the most beneficial habits you can form as a new programmer.

Cuando decidas si escribir o no un comentario, pregúntate si has tenido que considerar varios enfoques antes de llegar a una forma razonable de hacer que algo funcione; si es así, escribe un comentario sobre tu solución. Es mucho más fácil borrar más tarde los comentarios sobrantes que volver atrás y escribir comentarios para un programa escasamente comentado. A partir de ahora, utilizaré comentarios en los ejemplos de este libro para ayudar a explicar secciones de código.

When you're deciding whether to write a comment, ask yourself if you had to consider several approaches before coming up with a reasonable way to make something work; if so, write a comment about your solution. It's much easier to delete extra comments later than to go back and write comments for a sparsely commented program. From now on, I'll use comments in examples throughout this book to help explain sections of code.

Pruébalo tú mismo (*Try It Yourself*)

2-11. Añadir comentarios : Elige dos de los programas que has escrito, y añade al menos un comentario a cada uno. Si no tienes nada específico que escribir porque tus programas son demasiado sencillos en este momento, simplemente añade tu nombre y la fecha actual

en la parte superior de cada archivo de programa. Después escribe una frase describiendo lo que hace el programa.

2-11. Adding Comments : Choose two of the programs you've written, and add at least one comment to each. If you don't have anything specific to write because your programs are too simple at this point, just add your name and the current date at the top of each program file. Then write one sentence describing what the program does.

El Zen de Python (*The Zen of Python*)

Los programadores experimentados de Python te animarán a evitar la complejidad y a buscar la simplicidad siempre que sea posible. La filosofía de la comunidad Python está contenida en "El Zen de Python" de Tim Peters. Puedes acceder a este breve conjunto de principios para escribir buen código Python introduciendo `import this` en tu intérprete. No reproduciré aquí todo el "Zen de Python", pero compartiré algunas líneas para ayudarte a entender por qué deberían ser importantes para ti como programador principiante de Python.

Experienced Python programmers will encourage you to avoid complexity and aim for simplicity whenever possible. The Python community's philosophy is contained in "The Zen of Python" by Tim Peters. You can access this brief set of principles for writing good Python code by entering `import this` into your interpreter. I won't reproduce the entire "Zen of Python" here, but I'll share a few lines to help you understand why they should be important to you as a beginning Python programmer.

```
>>> import this
The Zen of Python, by Tim Peters
Beautiful is better than ugly.
```

Los programadores de Python abrazan la noción de que el código puede ser bello y elegante. En programación, la gente resuelve problemas. Los programadores siempre han respetado las soluciones a los problemas bien diseñadas, eficientes e incluso bellas. A medida que aprendas más sobre Python y lo utilices para escribir más código, alguien podría mirar un día por encima de tu hombro y decir: "¡Vaya, qué código más bonito!".

Python programmers embrace the notion that code can be beautiful and elegant. In programming, people solve problems. Programmers have always respected well-designed, efficient, and even beautiful solutions to problems. As you learn more about Python and use it to write more code, someone might look over your shoulder one day and say, “Wow, that’s some beautiful code!”

Simple is better than complex.

Si puedes elegir entre una solución simple y una compleja, y ambas funcionan, utiliza la solución simple. Tu código será más fácil de mantener, y será más fácil para ti y para otros construir sobre ese código más adelante.

If you have a choice between a simple and a complex solution, and both work, use the simple solution. Your code will be easier to maintain, and it will be easier for you and others to build on that code later on.

Complex is better than complicated.

La vida real es desordenada, y a veces una solución sencilla a un problema es inalcanzable. En ese caso, utiliza la solución más sencilla que funcione.

Real life is messy, and sometimes a simple solution to a problem is unattainable. In that case, use the simplest solution that works.

Readability counts.

Aunque tu código sea complejo, procura que sea legible. Cuando trabajes en un proyecto que implique una codificación compleja, céntrate en escribir comentarios informativos para ese código.

Even when your code is complex, aim to make it readable. When you’re working on a project that involves complex coding, focus on writing informative comments for that code.

There should be one-- and preferably only one --obvious way to do it.

Si se pide a dos programadores de Python que resuelvan el mismo problema, deberían llegar a soluciones bastante compatibles. Esto no quiere decir que no haya lugar para la creatividad en la programación. Al contrario, ¡hay mucho espacio para la creatividad! Sin embargo, gran parte de la programación consiste en utilizar pequeños enfoques comunes a situaciones sencillas dentro de un proyecto más amplio y creativo. Las tuercas y los tornillos de tus programas deben tener sentido para otros programadores de Python.

If two Python programmers are asked to solve the same problem, they should come up with fairly compatible solutions. This is not to say there's no room for creativity in programming. On the contrary, there is plenty of room for creativity! However, much of programming consists of using small, common approaches to simple situations within a larger, more creative project. The nuts and bolts of your programs should make sense to other Python programmers.

Now is better than never.

Podrías pasarte el resto de tu vida aprendiendo todos los entresijos de Python y de la programación en general, pero entonces nunca completarías ningún proyecto. No intentes escribir código perfecto; escribe código que funcione, y luego decide si mejoras tu código para ese proyecto o pasas a algo nuevo.

You could spend the rest of your life learning all the intricacies of Python and of programming in general, but then you'd never complete any projects. Don't try to write perfect code; write code that works, and then decide whether to improve your code for that project or move on to something new.

Cuando pases al siguiente capítulo y empieces a profundizar en temas más complicados, intenta mantener en mente esta filosofía de simplicidad y claridad. Los programadores experimentados respetarán más tu código y estarán encantados de darte su opinión y colaborar contigo en proyectos interesantes.

As you continue to the next chapter and start digging into more involved topics, try to keep this philosophy of simplicity and clarity in mind. Experienced programmers will respect your code more and will be happy to give you feedback and collaborate with you on interesting projects.

Pruébalo tú mismo (*Try It Yourself*)

2-12. Zen de Python: Entra en `import this` en una sesión de terminal de Python y hojear los principios adicionales.

2-12. Zen of Python: Enter `import this` into a Python terminal session and skim through the additional principles.

Resumen (*Summary*)

En este capítulo has aprendido a trabajar con variables. Aprendiste a utilizar nombres de variables descriptivos y a resolver errores de nombre y de sintaxis cuando surgen. Aprendiste qué son las cadenas y cómo mostrarlas utilizando minúsculas, mayúsculas y mayúsculas de título. Empezaste a utilizar los espacios en blanco para organizar la salida de forma ordenada, y aprendiste a eliminar los elementos innecesarios de una cadena. Empezaste a trabajar con números enteros y flotantes, y aprendiste algunas de las formas de trabajar con datos numéricos. También aprendiste a escribir comentarios explicativos para que tu código sea más fácil de leer para ti y para los demás. Por último, has leído sobre la filosofía de mantener tu código lo más simple posible, siempre que sea posible.

In this chapter you learned how to work with variables. You learned to use descriptive variable names and resolve name errors and syntax errors when they arise. You learned what strings are and how to display them using lowercase, uppercase, and title case. You started using whitespace to organize output neatly, and you learned how to remove unneeded elements from a string. You started working with integers and floats, and you learned some of the ways you can work with numerical data. You also learned to write explanatory comments to make your code easier for you and others to read. Finally, you read about the philosophy of keeping your code as simple as possible, whenever possible.

En el Capítulo 3 , aprenderás a almacenar colecciones de información en estructuras de datos llamadas *lists* . También aprenderás a trabajar a través de una lista, manipulando cualquier información de esa lista.

In Chapter 3 , you'll learn how to store collections of information in data structures called *lists* . You'll also learn how to work through a list, manipulating any information in that list.

Capítulo 3: Introducción a las listas (Chapter 3: Introducing Lists)

3

Introducción a las listas (3 Introducing Lists)



DESCRIPCIÓN DE LA
IMAGEN (En español):

La imagen muestra una ilustración de una serpiente envuelta alrededor de un motor a reacción. La serpiente tiene una expresión sonriente y lleva gafas protectoras. El motor está detallado con una especie de hélice al frente. El fondo de la imagen es de un tono gris, y el diseño está contenido en un círculo con borde oscuro.

En este capítulo y en el siguiente aprenderás qué son las listas y cómo empezar a trabajar con los elementos de una lista. Las listas te permiten almacenar conjuntos de información en un solo lugar, tanto si tienes unos pocos elementos como millones de ellos. Las listas son una de las funciones más potentes de Python, fácilmente accesibles para los nuevos programadores, y unen muchos conceptos importantes de la programación.

In this chapter and the next you'll learn what lists are and how to start working with the elements in a list. Lists allow you to store sets of information in one place, whether you have just a few items or millions of items. Lists are one of Python's most powerful features readily accessible to new programmers, and they tie together many important concepts in programming.

¿Qué es una lista? (What Is a List?)

Un *list* es una colección de elementos en un orden determinado. Puedes hacer una lista que incluya las letras del alfabeto, los dígitos del 0 al 9 o los nombres de todas las personas de tu familia. Puedes poner lo que quieras en una lista, y los elementos de tu lista no tienen

por qué estar relacionados de ninguna manera en particular. Como una lista suele contener más de un elemento, es buena idea que el nombre de tu lista sea plural, como `letters` , `digits` o `names` .

A *list* is a collection of items in a particular order. You can make a list that includes the letters of the alphabet, the digits from 0 to 9, or the names of all the people in your family. You can put anything you want into a list, and the items in your list don't have to be related in any particular way. Because a list usually contains more than one element, it's a good idea to make the name of your list plural, such as `letters` , `digits` , or `names` .

En Python, los corchetes (`[]`) indican una lista, y los elementos individuales de la lista se separan mediante comas. He aquí un ejemplo sencillo de una lista que contiene algunos tipos de bicicletas:

In Python, square brackets (`[]`) indicate a list, and individual elements in the list are separated by commas. Here's a simple example of a list that contains a few kinds of bicycles:

bicicletas.py

bicycles.py

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']  
print(bicycles)
```

Si pides a Python que imprima una lista, Python devuelve su representación de la lista, incluidos los corchetes:

If you ask Python to print a list, Python returns its representation of the list, including the square brackets:

```
['trek', 'cannondale', 'redline', 'specialized']
```

Como ésta no es la salida que quieres que vean tus usuarios, vamos a aprender cómo acceder a los elementos individuales de una lista.

Because this isn't the output you want your users to see, let's learn how to access the individual items in a list.

Acceder a los elementos de una lista (*Accessing Elements in a List*)

Las listas son colecciones ordenadas, por lo que puedes acceder a cualquier elemento de una lista indicando a Python la posición, o *index*, del elemento deseado. Para acceder a un elemento de una lista, escribe el nombre de la lista seguido del índice del elemento entre corchetes.

Lists are ordered collections, so you can access any element in a list by telling Python the position, or *index*, of the item desired. To access an element in a list, write the name of the list followed by the index of the item enclosed in square brackets.

Por ejemplo, vamos a sacar la primera bicicleta de la lista `bicycles` :

For example, let's pull out the first bicycle in the list `bicycles` :

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']  
print(bicycles[0])
```

Cuando pedimos un solo elemento de una lista, Python devuelve sólo ese elemento sin corchetes:

When we ask for a single item from a list, Python returns just that element without square brackets:

```
trek
```

Este es el resultado que quieres que vean tus usuarios: una salida limpia y bien formateada.

This is the result you want your users to see: clean, neatly formatted output.

También puedes utilizar los métodos de cadena del Capítulo 2 en cualquier elemento de esta lista. Por ejemplo, puedes formatear el elemento 'trek' para que tenga un aspecto más presentable utilizando el método `title()` :

You can also use the string methods from Chapter 2 on any element in this list. For example, you can format the element 'trek' to look more presentable by using the `title()` method:

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']  
print(bicycles[0].title())
```

Este ejemplo produce la misma salida que el ejemplo anterior, excepto que 'Trek' está en mayúsculas.

This example produces the same output as the preceding example, except 'Trek' is capitalized.

Las posiciones de índice empiezan en 0, no en 1 (*Index Positions Start at 0, Not 1*)

Python considera que el primer elemento de una lista está en la posición 0, no en la 1. Esto es así en la mayoría de los lenguajes de programación, y la razón tiene que ver con cómo se implementan las operaciones de lista en un nivel inferior. Si recibes resultados inesperados, pregúntate si estás cometiendo un error simple pero común de uno en uno.

Python considers the first item in a list to be at position 0, not position 1. This is true of most programming languages, and the reason has to do with how the list operations are implemented at a lower level. If you're receiving unexpected results, ask yourself if you're making a simple but common off-by-one error.

El segundo elemento de una lista tiene un índice de 1. Utilizando este sistema de conteo, puedes obtener cualquier elemento que desees de una lista restando uno a su posición en la lista. Por ejemplo, para acceder al cuarto elemento de una lista, solicitas el elemento situado en el índice 3.

The second item in a list has an index of 1. Using this counting system, you can get any element you want from a list by subtracting one from its position in the list. For instance, to access the fourth item in a list, you request the item at index 3.

A continuación se piden las bicicletas en los índices 1 y 3 :

The following asks for the bicycles at index 1 and index 3 :

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']  
print(bicycles[1])  
print(bicycles[3])
```

Este código devuelve la segunda y la cuarta bicicleta de la lista:

This code returns the second and fourth bicycles in the list:

```
cannondale  
specialized
```

Python tiene una sintaxis especial para acceder al último elemento de una lista. Si pides el elemento en el índice -1 , Python siempre devuelve el último elemento de la lista:

Python has a special syntax for accessing the last element in a list. If you ask for the item at index -1 , Python always returns the last item in the list:

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']  
print(bicycles[-1])
```

Este código devuelve el valor 'specialized' . Esta sintaxis es bastante útil, porque a menudo querrás acceder a los últimos elementos de una lista sin saber exactamente su longitud. Esta convención se extiende también a otros valores de índice negativos. El índice -2 devuelve el segundo elemento desde el final de la lista, el índice -3 devuelve el tercer elemento desde el final, y así sucesivamente.

This code returns the value 'specialized' . This syntax is quite useful, because you'll often want to access the last items in a list without knowing exactly how long the list is. This convention extends to other negative index values as well. The index -2 returns the second item from the end of the list, the index -3 returns the third item from the end, and so forth.

Utilizar valores individuales de una lista *(Using Individual Values from a List)*

Puedes utilizar valores individuales de una lista del mismo modo que utilizarías cualquier otra variable. Por ejemplo, puedes utilizar f-cadenas para crear un mensaje basado en un valor de una lista.

You can use individual values from a list just as you would any other variable. For example, you can use f-strings to create a message based on a value from a list.

Intentemos sacar la primera bicicleta de la lista y componer un mensaje utilizando ese valor:

Let's try pulling the first bicycle from the list and composing a message using that value:

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']  
message = f"My first bicycle was a {bicycles[0].title()}."  
  
print(message)
```

Construimos una frase utilizando el valor en `bicycles[0]` y lo asignamos a la variable `message` . El resultado es una frase sencilla sobre la primera bicicleta de la lista:

We build a sentence using the value at `bicycles[0]` and assign it to the variable `message` . The output is a simple sentence about the first bicycle in the list:

My first bicycle was a Trek.

Pruébalo tú mismo *(Try It Yourself)*

Prueba estos breves programas para tener alguna experiencia de primera mano con las listas de Python. Tal vez quieras crear una nueva carpeta para los ejercicios de cada capítulo, para mantenerlos organizados.

Try these short programs to get some firsthand experience with Python's lists. You might want to create a new folder for each chapter's exercises, to keep them organized.

3-1. Nombres: Almacena los nombres de algunos de tus amigos en una lista llamada `names` . Imprime el nombre de cada persona accediendo a cada elemento de la lista, de uno en uno.

3-1. Names: Store the names of a few of your friends in a list called `names` . Print each person's name by accessing each element in the list, one at a time.

3-2. Saludos: Empieza con la lista que utilizaste en el Ejercicio 3-1, pero en lugar de imprimir sólo el nombre de cada persona, imprime un mensaje para ellos. El texto de cada mensaje debe ser el mismo, pero cada mensaje debe estar personalizado con el nombre de la persona.

3-2. Greetings: Start with the list you used in Exercise 3-1, but instead of just printing each person's name, print a message to them. The text of each message should be the same, but each message should be personalized with the person's name.

3-3. Tu propia lista: Piensa en tu medio de transporte favorito, como una moto o un coche, y haz una lista que almacene varios ejemplos. Utiliza tu lista para imprimir una serie de afirmaciones sobre estos objetos, como "Me gustaría tener una moto Honda".

3-3. Your Own List: Think of your favorite mode of transportation, such as a motorcycle or a car, and make a list that stores several examples. Use your list to print a series of statements about these items, such as “I would like to own a Honda motorcycle.”

Modificar, añadir y eliminar elementos (Modifying, Adding, and Removing Elements)

La mayoría de las listas que crees serán *dynamic*, lo que significa que construirás una lista y luego añadirás y quitarás elementos de ella a medida que tu programa siga su curso. Por ejemplo, puedes crear un juego en el que un jugador tenga que disparar a unos alienígenas desde el cielo. Podrías almacenar el conjunto inicial de alienígenas en una lista y luego eliminar un alienígena de la lista cada vez que se derribe uno. Cada vez que aparezca un nuevo alienígena en la pantalla, lo añades a la lista. Tu lista de alienígenas aumentará y disminuirá de longitud a lo largo del juego.

Most lists you create will be *dynamic*, meaning you'll build a list and then add and remove elements from it as your program runs its course. For example, you might create a game in which a player has to shoot aliens out of the sky. You could store the initial set of aliens in a list and then remove an alien from the list each time one is shot down. Each time a new alien appears on the screen, you add it to the list. Your list of aliens will increase and decrease in length throughout the course of the game.

Modificar elementos de una lista (Modifying Elements in a List)

La sintaxis para modificar un elemento es similar a la sintaxis para acceder a un elemento de una lista. Para modificar un elemento, utiliza el nombre de la lista seguido del índice del elemento que quieres cambiar y, a continuación, proporciona el nuevo valor que quieres que tenga ese elemento.

The syntax for modifying an element is similar to the syntax for accessing an element in a list. To change an element, use the name of the list followed by the index of the element you want to change, and then provide the new value you want that item to have.

Por ejemplo, supongamos que tenemos una lista de motocicletas y que el primer elemento de la lista es 'honda'. Podemos cambiar el valor de este primer elemento una vez creada la lista:

For example, say we have a list of motorcycles and the first item in the list is 'honda' . We can change the value of this first item after the list has been created:

motos.py

motorcycles.py

```
motorcycles = ['honda', 'yamaha', 'suzuki']  
print(motorcycles)
```

```
motorcycles[0] = 'ducati'  
print(motorcycles)
```

Aquí definimos la lista `motorcycles` , con 'honda' como primer elemento. Luego cambiamos el valor del primer elemento a 'ducati' . La salida muestra que se ha cambiado el primer elemento, mientras que el resto de la lista permanece igual:

Here we define the list `motorcycles` , with 'honda' as the first element. Then we change the value of the first item to 'ducati' . The output shows that the first item has been changed, while the rest of the list stays the same:

```
['honda', 'yamaha', 'suzuki']  
['ducati', 'yamaha', 'suzuki']
```

Puedes cambiar el valor de cualquier elemento de una lista, no sólo del primero.

You can change the value of any item in a list, not just the first item.

Añadir elementos a una lista (*Adding Elements to a List*)

Puedes querer añadir un nuevo elemento a una lista por muchas razones. Por ejemplo, puedes querer que aparezcan nuevos alienígenas en un juego, añadir nuevos datos a una visualización o añadir nuevos usuarios registrados a un sitio web que hayas construido.

Python proporciona varias formas de añadir nuevos datos a las listas existentes.

You might want to add a new element to a list for many reasons. For example, you might want to make new aliens appear in a game, add new data to a visualization, or add new registered users to a website you've built. Python provides several ways to add new data to existing lists.

Añadir elementos al final de una lista (Appending Elements to the End of a List)

La forma más sencilla de añadir un nuevo elemento a una lista es *append*. Cuando añades un elemento a una lista, el nuevo elemento se añade al final de la lista. Utilizando la misma lista que teníamos en el ejemplo anterior, añadiremos el nuevo elemento 'ducati' al final de la lista:

The simplest way to add a new element to a list is to *append* the item to the list. When you append an item to a list, the new element is added to the end of the list. Using the same list we had in the previous example, we'll add the new element 'ducati' to the end of the list:

```
motorcycles = ['honda', 'yamaha', 'suzuki']  
print(motorcycles)
```

```
motorcycles.append('ducati')  
print(motorcycles)
```

Aquí el método `append()` añade 'ducati' al final de la lista, sin afectar a ninguno de los demás elementos de la lista:

Here the `append()` method adds 'ducati' to the end of the list, without affecting any of the other elements in the list:

```
['honda', 'yamaha', 'suzuki']  
['honda', 'yamaha', 'suzuki', 'ducati']
```

El método `append()` facilita la construcción dinámica de listas. Por ejemplo, puedes empezar con una lista vacía y luego añadir elementos a la lista mediante una serie de llamadas a `append()`. Utilizando una lista vacía, añadamos a la lista los elementos 'honda', 'yamaha', y 'suzuki':

The `append()` method makes it easy to build lists dynamically. For example, you can start with an empty list and then add items to the list using a series of `append()` calls. Using an empty list, let's add the elements 'honda' , 'yamaha' , and 'suzuki' to the list:

```
motorcycles = []

motorcycles.append('honda')
motorcycles.append('yamaha')
motorcycles.append('suzuki')

print(motorcycles)
```

La lista resultante tiene exactamente el mismo aspecto que las listas de los ejemplos anteriores:

The resulting list looks exactly the same as the lists in the previous examples:

```
['honda', 'yamaha', 'suzuki']
```

Construir listas de esta forma es muy habitual, porque a menudo no sabrás los datos que tus usuarios quieren almacenar en un programa hasta después de que éste se esté ejecutando. Para que tus usuarios tengan el control, empieza definiendo una lista vacía que contendrá los valores de los usuarios. A continuación, añade cada nuevo valor proporcionado a la lista que acabas de crear.

Building lists this way is very common, because you often won't know the data your users want to store in a program until after the program is running. To put your users in control, start by defining an empty list that will hold the users' values. Then append each new value provided to the list you just created.

Insertar elementos en una lista (Inserting Elements into a List)

Puedes añadir un nuevo elemento en cualquier posición de tu lista utilizando el método `insert()` . Para ello, especifica el índice del nuevo elemento y el valor del nuevo elemento:

You can add a new element at any position in your list by using the `insert()` method. You do this by specifying the index of the new element and the value of the new item:

```
motorcycles = ['honda', 'yamaha', 'suzuki']
```

```
motorcycles.insert(0, 'ducati')  
print(motorcycles)
```

En este ejemplo, insertamos el valor `'ducati'` al principio de la lista. El método `insert()` abre un espacio en la posición `0` y almacena el valor `'ducati'` en ese lugar:

In this example, we insert the value `'ducati'` at the beginning of the list. The `insert()` method opens a space at position `0` and stores the value `'ducati'` at that location:

```
['ducati', 'honda', 'yamaha', 'suzuki']
```

Esta operación desplaza todos los demás valores de la lista una posición hacia la derecha.

This operation shifts every other value in the list one position to the right.

Eliminar elementos de una lista *(Removing Elements from a List)*

A menudo, querrás eliminar un elemento o un conjunto de elementos de una lista. Por ejemplo, cuando un jugador derriba a un alienígena del cielo, lo más probable es que quieras eliminarlo de la lista de alienígenas activos. O cuando un usuario decide cancelar su cuenta en una aplicación web que has creado, querrás eliminar a ese usuario de la lista de usuarios activos. Puedes eliminar un elemento según su posición en la lista o según su valor.

Often, you'll want to remove an item or a set of items from a list. For example, when a player shoots down an alien from the sky, you'll most likely want to remove it from the list of active aliens. Or when a user decides to cancel their account on a web application you created, you'll want to remove that user from the list of active users. You can remove an item according to its position in the list or according to its value.

Eliminar un elemento utilizando la sentencia del (Removing an Item Using the del Statement)

Si conoces la posición del elemento que quieres eliminar de una lista, puedes utilizar la sentencia `del` :

If you know the position of the item you want to remove from a list, you can use the `del` statement:

```
motorcycles = ['honda', 'yamaha', 'suzuki']  
print(motorcycles)
```

```
del motorcycles[0]  
print(motorcycles)
```

Aquí utilizamos la sentencia `del` para eliminar el primer elemento, 'honda' , de la lista de motocicletas:

Here we use the `del` statement to remove the first item, 'honda' , from the list of motorcycles:

```
['honda', 'yamaha', 'suzuki']  
['yamaha', 'suzuki']
```

Puedes eliminar un elemento de cualquier posición de una lista utilizando la sentencia `del` si conoces su índice. Por ejemplo, así se elimina el segundo elemento, 'yamaha' , de la lista:

You can remove an item from any position in a list using the `del` statement if you know its index. For example, here's how to remove the second item, 'yamaha' , from the list:

```
motorcycles = ['honda', 'yamaha', 'suzuki']  
print(motorcycles)
```

```
del motorcycles[1]  
print(motorcycles)
```

El segundo moto se elimina de la lista:

The second motorcycle is deleted from the list:

```
['honda', 'yamaha', 'suzuki']
```

```
['honda', 'suzuki']
```

En ambos ejemplos, ya no puedes acceder al valor que se ha eliminado de la lista después de utilizar la sentencia `del`.

In both examples, you can no longer access the value that was removed from the list after the `del` statement is used.

Eliminar un elemento con el método `pop()` (Removing an Item Using the `pop()` Method)

A veces querrás utilizar el valor de un elemento después de eliminarlo de una lista. Por ejemplo, podrías querer obtener la posición *x* y *y* de un alienígena que acaba de ser derribado, para poder dibujar una explosión en esa posición. En una aplicación web, puede que quieras eliminar a un usuario de una lista de miembros activos y luego añadirlo a una lista de miembros inactivos.

Sometimes you'll want to use the value of an item after you remove it from a list. For example, you might want to get the *x* and *y* position of an alien that was just shot down, so you can draw an explosion at that position. In a web application, you might want to remove a user from a list of active members and then add that user to a list of inactive members.

El método `pop()` elimina el último elemento de una lista, pero te permite trabajar con ese elemento después de eliminarlo. El término *pop* proviene de pensar en una lista como en una pila de elementos y de quitar un elemento de la parte superior de la pila. En esta analogía, la parte superior de una pila corresponde al final de una lista.

The `pop()` method removes the last item in a list, but it lets you work with that item after removing it. The term *pop* comes from thinking of a list as a stack of items and popping one item off the top of the stack. In this analogy, the top of a stack corresponds to the end of a list.

Vamos a sacar una moto de la lista de motos:

Let's pop a motorcycle from the list of motorcycles:

```
❶ motorcycles = ['honda', 'yamaha', 'suzuki']  
print(motorcycles)  
  
❷ popped_motorcycle = motorcycles.pop()  
❸ print(motorcycles)  
❹ print(popped_motorcycle)
```

Empezamos definiendo e imprimiendo la lista `motorcycles` ❶ . A continuación, extraemos un valor de la lista y lo asignamos a la variable `popped_motorcycle` ❷ . Imprimimos la lista ❸ para mostrar que se ha eliminado un valor de la lista. A continuación, imprimimos el valor extraído ❹ para demostrar que aún tenemos acceso al valor eliminado.

We start by defining and printing the list `motorcycles` ❶ . Then we pop a value from the list, and assign that value to the variable `popped_motorcycle` ❷ . We print the list ❸ to show that a value has been removed from the list. Then we print the popped value ❹ to prove that we still have access to the value that was removed.

La salida muestra que el valor 'suzuki' fue eliminado del final de la lista y ahora está asignado a la variable `popped_motorcycle` :

The output shows that the value 'suzuki' was removed from the end of the list and is now assigned to the variable `popped_motorcycle` :

```
['honda', 'yamaha', 'suzuki']  
['honda', 'yamaha']  
suzuki
```

¿Qué utilidad puede tener este método `pop()` ? Imagina que las motos de la lista están almacenadas en orden cronológico, según la fecha en que las poseíamos. Si éste es el caso,

podemos utilizar el método `pop()` para imprimir una frase sobre la última moto que compramos:

How might this `pop()` method be useful? Imagine that the motorcycles in the list are stored in chronological order, according to when we owned them. If this is the case, we can use the `pop()` method to print a statement about the last motorcycle we bought:

```
motorcycles = ['honda', 'yamaha', 'suzuki']
```

```
last_owned = motorcycles.pop()
print(f"The last motorcycle I owned was a {last_owned.title()}.")
```

La salida es una simple frase sobre la última moto que hemos poseído:

The output is a simple sentence about the most recent motorcycle we owned:

The last motorcycle I owned was a Suzuki.

Eliminar elementos de cualquier posición de una lista (Popping Items from Any Position in a List)

Puedes utilizar `pop()` para eliminar un elemento de cualquier posición de una lista incluyendo entre paréntesis el índice del elemento que quieres eliminar:

You can use `pop()` to remove an item from any position in a list by including the index of the item you want to remove in parentheses:

```
motorcycles = ['honda', 'yamaha', 'suzuki']
```

```
first_owned = motorcycles.pop(0)
print(f"The first motorcycle I owned was a {first_owned.title()}.")
```

Empezamos eliminando la primera moto de la lista, y luego imprimimos un mensaje sobre esa moto. La salida es una simple frase que describe la primera moto que tuve:

We start by popping the first motorcycle in the list, and then we print a message about that motorcycle. The output is a simple sentence describing the first motorcycle I ever owned:

The first motorcycle I owned was a Honda.

Recuerda que cada vez que utilizas `pop()`, el elemento con el que trabajas deja de almacenarse en la lista.

Remember that each time you use `pop()`, the item you work with is no longer stored in the list.

Si no estás seguro de si utilizar la sentencia `del` o el método `pop()`, aquí tienes una forma sencilla de decidirte: cuando quieras eliminar un elemento de una lista y no utilizar ese elemento de ninguna forma, utiliza la sentencia `del`; si quieres utilizar un elemento mientras lo eliminas, utiliza el método `pop()`.

If you're unsure whether to use the `del` statement or the `pop()` method, here's a simple way to decide: when you want to delete an item from a list and not use that item in any way, use the `del` statement; if you want to use an item as you remove it, use the `pop()` method.

Eliminar un elemento por valor (Removing an Item by Value)

A veces no conocerás la posición del valor que quieres eliminar de una lista. Si sólo conoces el valor del elemento que quieres eliminar, puedes utilizar el método `remove()`.

Sometimes you won't know the position of the value you want to remove from a list. If you only know the value of the item you want to remove, you can use the `remove()` method.

Por ejemplo, supongamos que queremos eliminar el valor 'ducati' de la lista de motocicletas:

For example, say we want to remove the value 'ducati' from the list of motorcycles:

```
motorcycles = ['honda', 'yamaha', 'suzuki', 'ducati']  
print(motorcycles)
```

```
motorcycles.remove('ducati')  
print(motorcycles)
```

Aquí el método `remove()` le dice a Python que averigüe dónde aparece 'ducati' en la lista y elimine ese elemento:

Here the `remove()` method tells Python to figure out where 'ducati' appears in the list and remove that element:

```
['honda', 'yamaha', 'suzuki', 'ducati']  
['honda', 'yamaha', 'suzuki']
```

También puedes utilizar el método `remove()` para trabajar con un valor que se está eliminando de una lista. Eliminemos el valor 'ducati' e imprimamos una razón para eliminarlo de la lista:

You can also use the `remove()` method to work with a value that's being removed from a list. Let's remove the value 'ducati' and print a reason for removing it from the list:

```
❶ motorcycles = ['honda', 'yamaha', 'suzuki', 'ducati']  
print(motorcycles)  
  
❷ too_expensive = 'ducati'  
❸ motorcycles.remove(too_expensive)  
print(motorcycles)  
❹ print(f"\nA {too_expensive.title()} is too expensive for me.")
```

Tras definir la lista ❶, asignamos el valor 'ducati' a una variable llamada `too_expensive` ❷. A continuación, utilizamos esta variable para decirle a Python qué valor debe eliminar de la lista ❸. El valor 'ducati' se ha eliminado de la lista ❹ pero sigue siendo accesible a través de la variable `too_expensive`, lo que nos permite imprimir una declaración sobre por qué hemos eliminado 'ducati' de la lista de motos:

After defining the list ❶ , we assign the value 'ducati' to a variable called `too_expensive` ❷ . We then use this variable to tell Python which value to remove from the list ❸ . The value 'ducati' has been removed from the list ❹ but is still accessible through the variable `too_expensive` , allowing us to print a statement about why we removed 'ducati' from the list of motorcycles:

```
['honda', 'yamaha', 'suzuki', 'ducati']  
['honda', 'yamaha', 'suzuki']
```

A Ducati is too expensive for me.

Nota (Note)

El método `remove()` sólo elimina la primera aparición del valor que especifiques. Si existe la posibilidad de que el valor aparezca más de una vez en la lista, tendrás que utilizar un bucle para asegurarte de que se eliminan todas las apariciones del valor. Aprenderás a hacerlo en el Capítulo 7 .

The `remove()` method deletes only the first occurrence of the value you specify. If there's a possibility the value appears more than once in the list, you'll need to use a loop to make sure all occurrences of the value are removed. You'll learn how to do this in Chapter 7 .

Pruébalo tú mismo (Try It Yourself)

Los siguientes ejercicios son un poco más complejos que los del Capítulo 2 , pero te dan la oportunidad de utilizar las listas de todas las formas descritas.

The following exercises are a bit more complex than those in Chapter 2 , but they give you an opportunity to use lists in all of the ways described.

3-4. Lista de invitados: Si pudieras invitar a cenar a cualquier persona, viva o fallecida, ¿a quién invitarías? Haz una lista que incluya al menos a tres personas a las que te gustaría invitar a cenar. Después, utiliza tu lista para imprimir un mensaje dirigido a cada persona,

invitándola a cenar.

3-4. Guest List: If you could invite anyone, living or deceased, to dinner, who would you invite? Make a list that includes at least three people you'd like to invite to dinner. Then use your list to print a message to each person, inviting them to dinner.

3-5. Cambiar la lista de invitados: Te acabas de enterar de que uno de tus invitados no puede asistir a la cena, así que tienes que enviar un nuevo juego de invitaciones. Tendrás que pensar en otra persona a la que invitar.

3-5. Changing Guest List: You just heard that one of your guests can't make the dinner, so you need to send out a new set of invitations. You'll have to think of someone else to invite.

Listado en español:

- Empieza con tu programa del Ejercicio 3-4. Añade una llamada a `print()` al final de tu programa, indicando el nombre del invitado que no puede asistir.
- Modifica tu lista, sustituyendo el nombre del invitado que no puede venir por el nombre de la nueva persona a la que invitas.
- Imprime un segundo juego de mensajes de invitación, uno para cada persona que sigue en tu lista.

Listing in English:

- Start with your program from Exercise 3-4. Add a `print()` call at the end of your program, stating the name of the guest who can't make it.
- Modify your list, replacing the name of the guest who can't make it with the name of the new person you are inviting.
- Print a second set of invitation messages, one for each person who is still in your list.

3-6. Más invitados: Acabas de encontrar una mesa más grande para cenar, así que ahora hay más espacio disponible. Piensa en tres invitados más a los que invitar a cenar.

3-6. More Guests: You just found a bigger dinner table, so now more space is available. Think of three more guests to invite to dinner.

Listado en español:

- Empieza con tu programa del Ejercicio 3-4 o 3-5. Añade una llamada a `print()` al final de tu programa, informando a la gente de que has encontrado una mesa más grande.
- Utiliza `insert()` para añadir un nuevo invitado al principio de tu lista.
- Utiliza `insert()` para añadir un nuevo invitado a la mitad de tu lista.
- Utiliza `append()` para añadir un nuevo invitado al final de tu lista.
- Imprime un nuevo conjunto de mensajes de invitación, uno para cada persona de tu lista.

Listing in English:

- Start with your program from Exercise 3-4 or 3-5. Add a `print()` call to the end of your program, informing people that you found a bigger table.
- Use `insert()` to add one new guest to the beginning of your list.
- Use `insert()` to add one new guest to the middle of your list.
- Use `append()` to add one new guest to the end of your list.
- Print a new set of invitation messages, one for each person in your list.

3-7. Reducir la lista de invitados: Te acabas de enterar de que tu nueva mesa no llegará a tiempo para la cena, y ahora sólo tienes espacio para dos invitados.

3-7. Shrinking Guest List: You just found out that your new dinner table won't arrive in time for the dinner, and now you have space for only two guests.

Listado en español:

- Empieza con tu programa del Ejercicio 3-6. Añade una nueva línea que imprima un mensaje diciendo que sólo puedes invitar a dos personas a cenar.
- Utiliza `pop()` para eliminar invitados de tu lista de uno en uno hasta que sólo queden dos nombres en ella. Cada vez que elimines un nombre de tu lista, imprime un mensaje para esa persona haciéndole saber que lamentas no poder invitarla a cenar.
- Imprime un mensaje a cada una de las dos personas que siguen en tu lista, haciéndoles saber que siguen invitadas.
- Utiliza `del` para eliminar los dos últimos nombres de tu lista, de modo que tengas una lista vacía. Imprime tu lista para asegurarte de que realmente tienes una lista vacía al final de tu programa.

Listing in English:

- Start with your program from Exercise 3-6. Add a new line that prints a message saying that you can invite only two people for dinner.

- Use `pop()` to remove guests from your list one at a time until only two names remain in your list. Each time you pop a name from your list, print a message to that person letting them know you're sorry you can't invite them to dinner.
 - Print a message to each of the two people still on your list, letting them know they're still invited.
 - Use `del` to remove the last two names from your list, so you have an empty list. Print your list to make sure you actually have an empty list at the end of your program.
-

Organizar una lista (*Organizing a List*)

A menudo, tus listas se crearán en un orden impredecible, porque no siempre puedes controlar el orden en que tus usuarios proporcionan sus datos. Aunque esto es inevitable en la mayoría de las circunstancias, con frecuencia querrás presentar la información en un orden determinado. A veces querrás que conserve el orden original de tu lista, y otras querrás cambiarlo. Python proporciona varias formas diferentes de organizar tus listas, dependiendo de la situación.

Often, your lists will be created in an unpredictable order, because you can't always control the order in which your users provide their data. Although this is unavoidable in most circumstances, you'll frequently want to present your information in a particular order. Sometimes you'll want to preserve the original order of your list, and other times you'll want to change the original order. Python provides a number of different ways to organize your lists, depending on the situation.

Ordenar una lista permanentemente con el método `sort()` (*Sorting a List Permanently with the `sort()` Method*)

El método `sort()` de Python hace relativamente fácil ordenar una lista. Imagina que tenemos una lista de coches y queremos cambiar el orden de la lista para almacenarlos alfabéticamente. Para simplificar la tarea, supongamos que todos los valores de la lista están en minúsculas:

Python's `sort()` method makes it relatively easy to sort a list. Imagine we have a list of cars and want to change the order of the list to store them alphabetically. To keep the task simple, let's assume that all the values in the list are lowercase:

coches.py

cars.py

```
cars = ['bmw', 'audi', 'toyota', 'subaru']  
cars.sort()  
print(cars)
```

El método `sort()` cambia el orden de la lista de forma permanente. Los coches están ahora en orden alfabético, y nunca podremos volver al orden original:

The `sort()` method changes the order of the list permanently. The cars are now in alphabetical order, and we can never revert to the original order:

```
['audi', 'bmw', 'subaru', 'toyota']
```

También puedes ordenar esta lista en orden alfabético inverso pasando el argumento `reverse=True` al método `sort()`. El siguiente ejemplo ordena la lista de coches en orden alfabético inverso:

You can also sort this list in reverse-alphabetical order by passing the argument `reverse=True` to the `sort()` method. The following example sorts the list of cars in reverse-alphabetical order:

```
cars = ['bmw', 'audi', 'toyota', 'subaru']  
cars.sort(reverse=True)  
print(cars)
```

De nuevo, el orden de la lista se modifica permanentemente:

Again, the order of the list is permanently changed:

```
['toyota', 'subaru', 'bmw', 'audi']
```


Ordenar una lista temporalmente con la función sorted() (Sorting a List Temporarily with the sorted() Function)

Para mantener el orden original de una lista pero presentarla ordenada, puedes utilizar la función `sorted()`. La función `sorted()` te permite presentar tu lista en un orden determinado, pero no afecta al orden real de la lista.

To maintain the original order of a list but present it in a sorted order, you can use the `sorted()` function. The `sorted()` function lets you display your list in a particular order, but doesn't affect the actual order of the list.

Probemos esta función con la lista de coches.

Let's try this function on the list of cars.

```
cars = ['bmw', 'audi', 'toyota', 'subaru']
```

```
❶ print("Here is the original list:")  
print(cars)
```

```
❷ print("\nHere is the sorted list:")  
print(sorted(cars))
```

```
❸ print("\nHere is the original list again:")  
print(cars)
```

Primero imprimimos la lista en su orden original ❶ y luego en orden alfabético ❷. Después de mostrar la lista en el nuevo orden, mostramos que la lista sigue almacenada en su orden original ❸:

We first print the list in its original order ❶ and then in alphabetical order ❷. After the list is displayed in the new order, we show that the list is still stored in its original order ❸:

```
Here is the original list:  
['bmw', 'audi', 'toyota', 'subaru']
```

Here is the sorted list:

```
['audi', 'bmw', 'subaru', 'toyota']
```

❶ Here is the original list again:

```
['bmw', 'audi', 'toyota', 'subaru']
```

Observa que la lista sigue existiendo en su orden original ❶ después de utilizar la función `sorted()`. La función `sorted()` también puede aceptar un argumento `reverse=True` si quieres mostrar una lista en orden alfabético inverso.

Notice that the list still exists in its original order ❶ after the `sorted()` function has been used. The `sorted()` function can also accept a `reverse=True` argument if you want to display a list in reverse-alphabetical order.

Nota (Note)

Ordenar una lista alfabéticamente es un poco más complicado cuando todos los valores no están en minúsculas. Hay varias formas de interpretar las mayúsculas a la hora de determinar un orden de clasificación, y especificar el orden exacto puede ser más complejo de lo que queremos tratar en este momento. Sin embargo, la mayoría de los enfoques de la ordenación se basarán directamente en lo que has aprendido en esta sección.

Sorting a list alphabetically is a bit more complicated when all the values are not in lowercase. There are several ways to interpret capital letters when determining a sort order, and specifying the exact order can be more complex than we want to deal with at this time. However, most approaches to sorting will build directly on what you learned in this section.

Imprimir una lista en orden inverso (*Printing a List in Reverse Order*)

Para invertir el orden original de una lista, puedes utilizar el método `reverse()`. Si originalmente guardamos la lista de coches en orden cronológico según la fecha en que los tuvimos, podríamos reordenar fácilmente la lista en orden cronológico inverso:

To reverse the original order of a list, you can use the `reverse()` method. If we originally stored the list of cars in chronological order according to when we owned them, we could easily rearrange the list into reverse-chronological order:

```
cars = ['bmw', 'audi', 'toyota', 'subaru']  
print(cars)
```

```
cars.reverse()  
print(cars)
```

Observa que `reverse()` no ordena alfabéticamente hacia atrás; simplemente invierte el orden de la lista:

Notice that `reverse()` doesn't sort backward alphabetically; it simply reverses the order of the list:

```
['bmw', 'audi', 'toyota', 'subaru']  
['subaru', 'toyota', 'audi', 'bmw']
```

El método `reverse()` cambia el orden de una lista permanentemente, pero puedes volver al orden original en cualquier momento aplicando `reverse()` a la misma lista por segunda vez.

The `reverse()` method changes the order of a list permanently, but you can revert to the original order anytime by applying `reverse()` to the same list a second time.

Encontrar la longitud de una lista (*Finding the Length of a List*)

Puedes averiguar rápidamente la longitud de una lista utilizando la función `len()`. La lista de este ejemplo tiene cuatro elementos, por lo que su longitud es 4 :

You can quickly find the length of a list by using the `len()` function. The list in this example has four items, so its length is 4 :

```
>>> cars = ['bmw', 'audi', 'toyota', 'subaru']
>>> len(cars)
4
```

Te resultará útil `len()` cuando necesites identificar el número de alienígenas que aún quedan por derribar en un juego, determinar la cantidad de datos que tienes que gestionar en una visualización o averiguar el número de usuarios registrados en un sitio web, entre otras tareas.

You'll find `len()` useful when you need to identify the number of aliens that still need to be shot down in a game, determine the amount of data you have to manage in a visualization, or figure out the number of registered users on a website, among other tasks.

Nota (Note)

Python cuenta los elementos de una lista empezando por uno, por lo que no deberías encontrarte con errores de uno en uno al determinar la longitud de una lista.

Python counts the items in a list starting with one, so you shouldn't run into any off-by-one errors when determining the length of a list.

Pruébalo tú mismo (Try It Yourself)

3-8. Ver el mundo: Piensa en al menos cinco lugares del mundo que te gustaría visitar.

3-8. Seeing the World: Think of at least five places in the world you'd like to visit.

Listado en español:

- Guarda los lugares en una lista. Asegúrate de que la lista no está en orden alfabético.
- Imprime tu lista en su orden original. No te preocupes por imprimir la lista ordenadamente; simplemente imprímela como una lista Python sin procesar.

- Utiliza `sorted()` para imprimir tu lista en orden alfabético sin modificar la lista real.
- Demuestra que tu lista sigue en su orden original imprimiéndola.
- Utiliza `sorted()` para imprimir tu lista en orden alfabético inverso sin modificar el orden de la lista original.
- Demuestra que tu lista sigue en el orden original imprimiéndola de nuevo.
- Utiliza `reverse()` para cambiar el orden de tu lista. Imprime la lista para mostrar que su orden ha cambiado.
- Utiliza `reverse()` para volver a cambiar el orden de la lista. Imprime la lista para mostrar que ha vuelto a su orden original.
- Utiliza `sort()` para cambiar tu lista de modo que se almacene en orden alfabético. Imprime la lista para mostrar que ha cambiado su orden.
- Utiliza `sort()` para cambiar tu lista de modo que se almacene en orden alfabético inverso. Imprime la lista para mostrar que su orden ha cambiado.

Listing in English:

- Store the locations in a list. Make sure the list is not in alphabetical order.
- Print your list in its original order. Don't worry about printing the list neatly; just print it as a raw Python list.
- Use `sorted()` to print your list in alphabetical order without modifying the actual list.
- Show that your list is still in its original order by printing it.
- Use `sorted()` to print your list in reverse-alphabetical order without changing the order of the original list.
- Show that your list is still in its original order by printing it again.
- Use `reverse()` to change the order of your list. Print the list to show that its order has changed.
- Use `reverse()` to change the order of your list again. Print the list to show it's back to its original order.
- Use `sort()` to change your list so it's stored in alphabetical order. Print the list to show that its order has been changed.
- Use `sort()` to change your list so it's stored in reverse-alphabetical order. Print the list to show that its order has changed.

3-9. Invitados a cenar: Trabajando con uno de los programas de los Ejercicios 3-4 a 3-7 (páginas 41-42), utiliza `len()` para imprimir un mensaje indicando el número de personas a las que invitas a cenar.

3-9. Dinner Guests: Working with one of the programs from Exercises 3-4 through 3-7 (pages 41–42), use `len()` to print a message indicating the number of people you're inviting to dinner.

3-10. Cada Función: Piensa en cosas que podrías almacenar en una lista. Por ejemplo, podrías hacer una lista de montañas, ríos, países, ciudades, idiomas o cualquier otra cosa que te apetezca. Escribe un programa que cree una lista que contenga estos elementos y luego utiliza cada función presentada en este capítulo al menos una vez.

3-10. Every Function: Think of things you could store in a list. For example, you could make a list of mountains, rivers, countries, cities, languages, or anything else you'd like. Write a program that creates a list containing these items and then uses each function introduced in this chapter at least once.

Evitar errores de índice al trabajar con listas (Avoiding Index Errors When Working with Lists)

Hay un tipo de error que es habitual ver cuando trabajas con listas por primera vez. Supongamos que tienes una lista con tres elementos, y preguntas por el cuarto elemento:

There's one type of error that's common to see when you're working with lists for the first time. Let's say you have a list with three items, and you ask for the fourth item:

motos.py

motorcycles.py

```
motorcycles = ['honda', 'yamaha', 'suzuki']  
print(motorcycles[3])
```

Este ejemplo da como resultado *index error* :

This example results in an *index error* :

Traceback (most recent call last):

```
File "motorcycles.py", line 2, in <module>
    print(motorcycles[3])
```

```
~~~~~^^^
```

IndexError: list index out of range

Python intenta darte el elemento en el índice 3. Pero cuando busca en la lista, ningún elemento de `motorcycles` tiene el índice 3. Debido a la naturaleza de la indexación en las listas, este error es típico. La gente cree que el tercer elemento es el número 3, porque empieza a contar en 1. Pero en Python el tercer elemento es el número 2, porque empieza a indexar en 0.

Python attempts to give you the item at index 3. But when it searches the list, no item in `motorcycles` has an index of 3. Because of the off-by-one nature of indexing in lists, this error is typical. People think the third item is item number 3, because they start counting at 1. But in Python the third item is number 2, because it starts indexing at 0.

Un error de índice significa que Python no puede encontrar un elemento en el índice que has solicitado. Si se produce un error de índice en tu programa, intenta ajustar en uno el índice que pides. Luego vuelve a ejecutar el programa para ver si los resultados son correctos.

An index error means Python can't find an item at the index you requested. If an index error occurs in your program, try adjusting the index you're asking for by one. Then run the program again to see if the results are correct.

Ten en cuenta que siempre que quieras acceder al último elemento de una lista, debes utilizar el índice `-1`. Esto siempre funcionará, aunque tu lista haya cambiado de tamaño desde la última vez que accediste a ella:

Keep in mind that whenever you want to access the last item in a list, you should use the index `-1`. This will always work, even if your list has changed size since the last time you accessed it:

```
motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles[-1])
```

El índice -1 siempre devuelve el último elemento de una lista, en este caso el valor 'suzuki' :

The index -1 always returns the last item in a list, in this case the value 'suzuki' :

suzuki

La única vez que este método provocará un error es cuando solicites el último elemento de una lista vacía:

The only time this approach will cause an error is when you request the last item from an empty list:

```
motorcycles = []  
print(motorcycles[-1])
```

No hay ningún elemento en `motorcycles` , por lo que Python devuelve otro error de índice:

No items are in `motorcycles` , so Python returns another index error:

Traceback (most recent call last):

```
File "motorcycles.py", line 3, in <module>  
    print(motorcycles[-1])  
~~~~~  
~~~~~
```

IndexError: list index out of range

Si se produce un error de índice y no sabes cómo resolverlo, intenta imprimir tu lista o simplemente imprimir la longitud de tu lista. Tu lista puede tener un aspecto muy distinto del que pensabas, sobre todo si ha sido gestionada dinámicamente por tu programa. Ver la lista real, o el número exacto de elementos de tu lista, puede ayudarte a resolver esos errores lógicos.

If an index error occurs and you can't figure out how to resolve it, try printing your list or just printing the length of your list. Your list might look much different than you thought it

did, especially if it has been managed dynamically by your program. Seeing the actual list, or the exact number of items in your list, can help you sort out such logical errors.

Pruébalo tú mismo *(Try It Yourself)*

3-11. Error intencionado: Si todavía no has recibido un error de índice en uno de tus programas, intenta que se produzca uno. Cambia un índice en uno de tus programas para producir un error de índice. Asegúrate de corregir el error antes de cerrar el programa.

3-11. Intentional Error: If you haven't received an index error in one of your programs yet, try to make one happen. Change an index in one of your programs to produce an index error. Make sure you correct the error before closing the program.

Resumen *(Summary)*

En este capítulo has aprendido qué son las listas y cómo trabajar con los elementos individuales de una lista. Aprendiste a definir una lista y a añadir y eliminar elementos. Aprendiste a ordenar listas de forma permanente y temporal para su visualización. También aprendiste a averiguar la longitud de una lista y a evitar errores de índice cuando trabajas con listas.

In this chapter, you learned what lists are and how to work with the individual items in a list. You learned how to define a list and how to add and remove elements. You learned how to sort lists permanently and temporarily for display purposes. You also learned how to find the length of a list and how to avoid index errors when you're working with lists.

En el Capítulo 4 aprenderás a trabajar con los elementos de una lista de forma más eficaz. Recorriendo en bucle cada uno de los elementos de una lista con unas pocas líneas de código podrás trabajar con eficacia, incluso cuando tu lista contenga miles o millones de elementos.

In Chapter 4 you'll learn how to work with items in a list more efficiently. By looping through each item in a list using just a few lines of code you'll be able to work efficiently, even when your list contains thousands or millions of items.

Capítulo 4: Trabajar con listas (Chapter 4: Working with Lists)

4

Trabajar con listas (4 Working with Lists)



DESCRIPCIÓN DE LA
IMAGEN (En español):

La imagen muestra una ilustración en blanco y negro de una serpiente sonriente que lleva enrollado un motor de avión. La serpiente tiene un diseño estilizado y dinámico, con su cuerpo alrededor del motor que parece estar a punto de despegar. No se aprecia texto alguno en la imagen, solo el dibujo que abarca todo el espacio circular.

En el Capítulo 3 aprendiste a hacer una lista simple y a trabajar con los elementos individuales de una lista. En este capítulo aprenderás a recorrer en bucle una lista entera utilizando sólo unas pocas líneas de código, independientemente de lo larga que sea la lista. *Looping* te permite realizar la misma acción, o conjunto de acciones, con cada elemento de una lista. Como resultado, podrás trabajar eficazmente con listas de cualquier longitud, incluidas las que tienen miles o incluso millones de elementos.

In Chapter 3 you learned how to make a simple list, and you learned to work with the individual elements in a list. In this chapter you'll learn how to loop through an entire list using just a few lines of code, regardless of how long the list is. *Looping* allows you to take the same action, or set of actions, with every item in a list. As a result, you'll be able to work efficiently with lists of any length, including those with thousands or even millions of items.

Recorrer en bucle toda una lista (*Looping Through an Entire List*)

A menudo querrás recorrer todas las entradas de una lista, realizando la misma tarea con cada elemento. Por ejemplo, en un juego puedes querer mover todos los elementos de la pantalla la misma cantidad. En una lista de números, quizá quieras realizar la misma

operación estadística con cada elemento. O quizá quieras mostrar cada titular de una lista de artículos de un sitio web. Cuando quieras realizar la misma acción con cada elemento de una lista, puedes utilizar el bucle `for` de Python.

You'll often want to run through all entries in a list, performing the same task with each item. For example, in a game you might want to move every element on the screen by the same amount. In a list of numbers, you might want to perform the same statistical operation on every element. Or perhaps you'll want to display each headline from a list of articles on a website. When you want to do the same action with every item in a list, you can use Python's `for` loop.

Supongamos que tenemos una lista de nombres de magos y queremos imprimir cada nombre de la lista. Podríamos hacerlo recuperando cada nombre de la lista individualmente, pero este enfoque podría causar varios problemas. Por un lado, sería repetitivo hacerlo con una lista larga de nombres. Además, tendríamos que cambiar nuestro código cada vez que cambiara la longitud de la lista. Utilizar un bucle `for` evita estos dos problemas al dejar que Python gestione estas cuestiones internamente.

Say we have a list of magicians' names, and we want to print out each name in the list. We could do this by retrieving each name from the list individually, but this approach could cause several problems. For one, it would be repetitive to do this with a long list of names. Also, we'd have to change our code each time the list's length changed. Using a `for` loop avoids both of these issues by letting Python manage these issues internally.

Utilicemos un bucle `for` para imprimir cada nombre de una lista de magos:

Let's use a `for` loop to print out each name in a list of magicians:

magos.py

magicians.py

```
magicians = ['alice', 'david', 'carolina']  
for magician in magicians:  
    print(magician)
```

Empezamos definiendo una lista, como hicimos en el Capítulo 3 . Luego definimos un bucle `for` . Esta línea le dice a Python que saque un nombre de la lista `magicians` , y lo asocie a la variable `magician` . A continuación, le decimos a Python que imprima el nombre que se acaba de asignar a `magician` . A continuación, Python repite estas dos últimas líneas, una vez por cada nombre de la lista. Puede ayudar leer este código como "Para cada mago de la lista de magos, imprime el nombre del mago". La salida es una simple impresión de cada nombre de la lista:

We begin by defining a list, just as we did in Chapter 3 . Then we define a `for` loop. This line tells Python to pull a name from the list `magicians` , and associate it with the variable `magician` . Next, we tell Python to print the name that's just been assigned to `magician` . Python then repeats these last two lines, once for each name in the list. It might help to read this code as "For every magician in the list of `magicians`, print the magician's name." The output is a simple printout of each name in the list:

```
alice  
david  
carolina
```

Un vistazo más de cerca a los bucles *(A Closer Look at Looping)*

Los bucles son importantes porque son una de las formas más comunes que tiene un ordenador de automatizar tareas repetitivas. Por ejemplo, en un bucle simple como el que utilizamos en *magicians.py* , Python lee inicialmente la primera línea del bucle:

Looping is important because it's one of the most common ways a computer automates repetitive tasks. For example, in a simple loop like we used in *magicians.py* , Python initially reads the first line of the loop:

```
for magician in magicians:
```

Esta línea le dice a Python que recupere el primer valor de la lista `magicians` y lo asocie a la variable `magician` . Este primer valor es `'alice'` . A continuación, Python lee la siguiente línea:

This line tells Python to retrieve the first value from the list `magicians` and associate it with the variable `magician` . This first value is `'alice'` . Python then reads the next line:

```
print(magician)
```

Python imprime el valor actual de `magician` , que sigue siendo `'alice'` . Como la lista contiene más valores, Python vuelve a la primera línea del bucle:

Python prints the current value of `magician` , which is still `'alice'` . Because the list contains more values, Python returns to the first line of the loop:

```
for magician in magicians:
```

Python recupera el siguiente nombre de la lista, `'david'` , y asocia ese valor a la variable `magician` . A continuación, Python ejecuta la línea:

Python retrieves the next name in the list, `'david'` , and associates that value with the variable `magician` . Python then executes the line:

```
print(magician)
```

Python vuelve a imprimir el valor actual de `magician` , que ahora es `'david'` . Python repite todo el bucle una vez más con el último valor de la lista, `'carolina'` . Como no hay más valores en la lista, Python pasa a la siguiente línea del programa. En este caso no viene nada después del bucle `for` , por lo que el programa termina.

Python prints the current value of `magician` again, which is now `'david'` . Python repeats the entire loop once more with the last value in the list, `'carolina'` . Because no more values are in the list, Python moves on to the next line in the program. In this case nothing comes after the `for` loop, so the program ends.

Cuando utilices bucles por primera vez, ten en cuenta que el conjunto de pasos se repite una vez para cada elemento de la lista, sin importar cuántos elementos haya en ella. Si tienes un millón de elementos en tu lista, Python repite estos pasos un millón de veces, y normalmente muy rápido.

When you're using loops for the first time, keep in mind that the set of steps is repeated once for each item in the list, no matter how many items are in the list. If you have a million items in your list, Python repeats these steps a million times—and usually very quickly.

Ten en cuenta también, cuando escribas tus propios bucles `for`, que puedes elegir el nombre que quieras para la variable temporal que se asociará a cada valor de la lista. Sin embargo, es útil elegir un nombre significativo que represente un único elemento de la lista. Por ejemplo, ésta es una buena forma de iniciar un bucle `for` para una lista de gatos, una lista de perros y una lista general de elementos:

Also keep in mind when writing your own `for` loops that you can choose any name you want for the temporary variable that will be associated with each value in the list. However, it's helpful to choose a meaningful name that represents a single item from the list. For example, here's a good way to start a `for` loop for a list of cats, a list of dogs, and a general list of items:

```
for cat in cats:  
for dog in dogs:  
for item in list_of_items:
```

Estas convenciones de nomenclatura pueden ayudarte a seguir la acción que se realiza en cada elemento dentro de un bucle `for`. Utilizar nombres en singular y plural puede ayudarte a identificar si una sección de código está trabajando con un único elemento de la lista o con toda la lista.

These naming conventions can help you follow the action being done on each item within a `for` loop. Using singular and plural names can help you identify whether a section of code is working with a single element from the list or the entire list.

Hacer más trabajo dentro de un bucle `for` (*Doing More Work Within a for Loop*)

Puedes hacer prácticamente cualquier cosa con cada elemento de un bucle `for`. Continuemos con el ejemplo anterior imprimiendo un mensaje a cada mago, diciéndole que ha hecho un gran truco:

You can do just about anything with each item in a `for` loop. Let's build on the previous example by printing a message to each magician, telling them that they performed a great trick:

magos.py

magicians.py

```
magicians = ['alice', 'david', 'carolina']
for magician in magicians:
    print(f'{magician.title()}, that was a great trick!')
```

La única diferencia en este código es que componemos un mensaje para cada mago, empezando por su nombre. La primera vez que se pasa por el bucle, el valor de `magician` es `'alice'`, por lo que Python inicia el primer mensaje con el nombre `'Alice'`. La segunda vez, el mensaje empezará por `'David'`, y la tercera, por `'Carolina'`.

The only difference in this code is where we compose a message to each magician, starting with that magician's name. The first time through the loop the value of `magician` is `'alice'`, so Python starts the first message with the name `'Alice'`. The second time through, the message will begin with `'David'`, and the third time through, the message will begin with `'Carolina'`.

La salida muestra un mensaje personalizado para cada mago de la lista:

The output shows a personalized message for each magician in the list:

```
Alice, that was a great trick!
David, that was a great trick!
Carolina, that was a great trick!
```

También puedes escribir tantas líneas de código como quieras en el bucle `for`. Cada línea sangrada que sigue a la línea `for magician in magicians` se considera *inside the loop*, y cada línea sangrada se ejecuta una vez por cada valor de la lista. Por tanto, puedes hacer todo el trabajo que quieras con cada valor de la lista.

You can also write as many lines of code as you like in the `for` loop. Every indented line following the line `for magician in magicians` is considered *inside the loop*, and each indented line is executed once for each value in the list. Therefore, you can do as much work as you like with each value in the list.

Añadamos una segunda línea a nuestro mensaje, diciéndole a cada mago que esperamos su próximo truco:

Let's add a second line to our message, telling each magician that we're looking forward to their next trick:

```
magicians = ['alice', 'david', 'carolina']
for magician in magicians:
    print(f'{magician.title()}, that was a great trick!')
    print(f'I can't wait to see your next trick, {magician.title()}.\n')
```

Como hemos sangrado ambas llamadas a `print()`, cada línea se ejecutará una vez por cada mago de la lista. La nueva línea (`"\n"`) en la segunda llamada a `print()` inserta una línea en blanco después de cada pasada por el bucle. Esto crea un conjunto de mensajes agrupados ordenadamente para cada persona de la lista:

Because we have indented both calls to `print()`, each line will be executed once for every magician in the list. The newline (`"\n"`) in the second `print()` call inserts a blank line after each pass through the loop. This creates a set of messages that are neatly grouped for each person in the list:

```
Alice, that was a great trick!
I can't wait to see your next trick, Alice.
```

```
David, that was a great trick!
I can't wait to see your next trick, David.
```

```
Carolina, that was a great trick!
I can't wait to see your next trick, Carolina.
```

Puedes utilizar tantas líneas como quieras en tus bucles `for` . En la práctica, a menudo te resultará útil realizar varias operaciones diferentes con cada elemento de una lista cuando utilices un bucle `for` .

You can use as many lines as you like in your `for` loops. In practice, you'll often find it useful to do a number of different operations with each item in a list when you use a `for` loop.

Hacer algo después de un bucle `for` (*Doing Something After a `for` Loop*)

¿Qué ocurre una vez que un bucle `for` ha terminado de ejecutarse? Normalmente, querrás resumir un bloque de salida o pasar a otro trabajo que tu programa deba realizar.

What happens once a `for` loop has finished executing? Usually, you'll want to summarize a block of output or move on to other work that your program must accomplish.

Cualquier línea de código después del bucle `for` que no esté indentada se ejecuta una vez sin repetirse. Escribamos un mensaje de agradecimiento al grupo de magos en su conjunto, dándoles las gracias por haber ofrecido un espectáculo excelente. Para mostrar este mensaje de grupo después de que se hayan impreso todos los mensajes individuales, colocamos el mensaje de agradecimiento después del bucle `for` , sin sangría:

Any lines of code after the `for` loop that are not indented are executed once without repetition. Let's write a thank you to the group of magicians as a whole, thanking them for putting on an excellent show. To display this group message after all of the individual messages have been printed, we place the thank you message after the `for` loop, without indentation:

```
magicians = ['alice', 'david', 'carolina']
for magician in magicians:
    print(f'{magician.title()}, that was a great trick!')
    print(f'I can't wait to see your next trick, {magician.title()}.\n")

print("Thank you, everyone. That was a great magic show!")
```

Las dos primeras llamadas a `print()` se repiten una vez para cada mago de la lista, como has visto antes. Sin embargo, como la última línea no tiene sangría, sólo se imprime una vez:

The first two calls to `print()` are repeated once for each magician in the list, as you saw earlier. However, because the last line is not indented, it's printed only once:

Alice, that was a great trick!
I can't wait to see your next trick, Alice.

David, that was a great trick!
I can't wait to see your next trick, David.

Carolina, that was a great trick!
I can't wait to see your next trick, Carolina.

Thank you, everyone. That was a great magic show!

Cuando proceses datos utilizando un bucle `for`, verás que ésta es una buena forma de resumir una operación realizada en un conjunto de datos completo. Por ejemplo, puedes utilizar un bucle `for` para inicializar un juego recorriendo una lista de personajes y mostrando cada uno de ellos en la pantalla. Después de este bucle, podrías escribir un código adicional que muestre un botón *Play Now* una vez que todos los personajes se hayan dibujado en la pantalla.

When you're processing data using a `for` loop, you'll find that this is a good way to summarize an operation that was performed on an entire dataset. For example, you might use a `for` loop to initialize a game by running through a list of characters and displaying each character on the screen. You might then write some additional code after this loop that displays a *Play Now* button after all the characters have been drawn to the screen.

Evitar errores de sangría (*Avoiding Indentation Errors*)

Python utiliza la sangría para determinar cómo se relaciona una línea, o un grupo de líneas, con el resto del programa. En los ejemplos anteriores, las líneas que imprimían mensajes a los magos individuales formaban parte del bucle `for` porque estaban indentadas. El uso de la sangría en Python hace que el código sea muy fácil de leer. Básicamente, utiliza los espacios en blanco para obligarte a escribir código con un formato ordenado y una estructura visual clara. En los programas Python más largos, verás bloques de código

sangrados en diferentes niveles. Estos niveles de sangría te ayudan a tener una idea general de la organización del programa.

Python uses indentation to determine how a line, or group of lines, is related to the rest of the program. In the previous examples, the lines that printed messages to individual magicians were part of the `for` loop because they were indented. Python's use of indentation makes code very easy to read. Basically, it uses whitespace to force you to write neatly formatted code with a clear visual structure. In longer Python programs, you'll notice blocks of code indented at a few different levels. These indentation levels help you gain a general sense of the overall program's organization.

Cuando empieces a escribir código que dependa de una indentación adecuada, tendrás que estar atento a algunos errores comunes *indentation errors*. Por ejemplo, la gente a veces indenta líneas de código que no necesitan ser indentadas u olvida indentar líneas que necesitan ser indentadas. Ver ahora ejemplos de estos errores te ayudará a evitarlos en el futuro y a corregirlos cuando aparezcan en tus propios programas.

As you begin to write code that relies on proper indentation, you'll need to watch for a few common *indentation errors*. For example, people sometimes indent lines of code that don't need to be indented or forget to indent lines that need to be indented. Seeing examples of these errors now will help you avoid them in the future and correct them when they do appear in your own programs.

Examinemos algunos de los errores de sangría más comunes.

Let's examine some of the more common indentation errors.

Olvidar la sangría (*Forgetting to Indent*)

Sangrar siempre la línea que sigue a la sentencia `for` en un bucle. Si te olvidas, Python te lo recordará:

Always indent the line after the `for` statement in a loop. If you forget, Python will remind you:

magos.py

magicians.py

```
magicians = ['alice', 'david', 'carolina']  
for magician in magicians:  
    ❶ print(magician)
```

La llamada a `print()` ❶ debería estar indentada, pero no lo está. Cuando Python espera un bloque indentado y no lo encuentra, te indica con qué línea ha tenido un problema:

The call to `print()` ❶ should be indented, but it's not. When Python expects an indented block and doesn't find one, it lets you know which line it had a problem with:

```
File "magicians.py", line 3  
    print(magician)  
    ^
```

`IndentationError: expected an indented block after 'for' statement on line 2`

Normalmente puedes resolver este tipo de error de sangría sangrando la línea o líneas inmediatamente posteriores a la sentencia `for` .

You can usually resolve this kind of indentation error by indenting the line or lines immediately after the `for` statement.

Olvidar sangrar las líneas adicionales (*Forgetting to Indent Additional Lines*)

A veces tu bucle se ejecutará sin errores, pero no producirá el resultado esperado. Esto puede ocurrir cuando intentas realizar varias tareas en un bucle y te olvidas de sangrar algunas de sus líneas.

Sometimes your loop will run without any errors but won't produce the expected result. This can happen when you're trying to do several tasks in a loop and you forget to indent some of its lines.

Por ejemplo, esto es lo que ocurre cuando olvidamos sangrar la segunda línea del bucle que dice a cada mago que esperamos su próximo truco:

For example, this is what happens when we forget to indent the second line in the loop that tells each magician we're looking forward to their next trick:

```
magicians = ['alice', 'david', 'carolina']
for magician in magicians:
    print(f'{magician.title()}, that was a great trick!')
❶ print(f'I can't wait to see your next trick, {magician.title()}.\n")
```

Se supone que la segunda llamada a `print()` ❶ debe tener sangría, pero como Python encuentra al menos una línea con sangría después de la sentencia `for`, no informa de un error. Como resultado, la primera llamada a `print()` se ejecuta una vez por cada nombre de la lista porque está indentada. La segunda llamada a `print()` no está indentada, por lo que sólo se ejecuta una vez después de que el bucle haya terminado de ejecutarse. Como el valor final asociado a `magician` es `'carolina'`, ella es la única que recibe el mensaje "esperando el próximo truco":

The second call to `print()` ❶ is supposed to be indented, but because Python finds at least one indented line after the `for` statement, it doesn't report an error. As a result, the first `print()` call is executed once for each name in the list because it is indented. The second `print()` call is not indented, so it is executed only once after the loop has finished running. Because the final value associated with `magician` is `'carolina'`, she is the only one who receives the "looking forward to the next trick" message:

```
Alice, that was a great trick!
David, that was a great trick!
Carolina, that was a great trick!
I can't wait to see your next trick, Carolina.
```

Esto es un *logical error*. La sintaxis es código Python válido, pero el código no produce el resultado deseado porque se produce un problema en su lógica. Si esperas que una determinada acción se repita una vez para cada elemento de una lista y sólo se ejecuta una

vez, determina si necesitas simplemente sangrar una línea o un grupo de líneas.

This is a *logical error* . The syntax is valid Python code, but the code does not produce the desired result because a problem occurs in its logic. If you expect to see a certain action repeated once for each item in a list and it's executed only once, determine whether you need to simply indent a line or a group of lines.

Sangrar innecesariamente (*Indenting Unnecessarily*)

Si sangras accidentalmente una línea que no necesita ser sangrada, Python te informa de la sangría inesperada:

If you accidentally indent a line that doesn't need to be indented, Python informs you about the unexpected indent:

hola_mundo.py

hello_world.py

```
message = "Hello Python world!"  
    print(message)
```

No necesitamos sangrar la llamada a `print()` , porque no forma parte de un bucle; por eso, Python informa de ese error:

We don't need to indent the `print()` call, because it isn't part of a loop; hence, Python reports that error:

```
File "hello_world.py", line 2  
    print(message)  
    ^  
IndentationError: unexpected indent
```

Puedes evitar los errores de sangrado inesperado sangrando sólo cuando tengas una razón específica para hacerlo. En los programas que estás escribiendo en este momento, las únicas líneas que debes sangrar son las acciones que quieres repetir para cada elemento de un bucle `for` .

You can avoid unexpected indentation errors by indenting only when you have a specific reason to do so. In the programs you're writing at this point, the only lines you should indent are the actions you want to repeat for each item in a `for` loop.

Sangrar innecesariamente después del bucle (*Indenting Unnecessarily After the Loop*)

Si accidentalmente indentas código que debería ejecutarse después de que haya terminado un bucle, ese código se repetirá una vez para cada elemento de la lista. A veces esto hace que Python informe de un error, pero a menudo provocará un error lógico.

If you accidentally indent code that should run after a loop has finished, that code will be repeated once for each item in the list. Sometimes this prompts Python to report an error, but often this will result in a logical error.

Por ejemplo, veamos qué ocurre si accidentalmente aplicamos una sangría a la línea que agradece a los magos en grupo que hayan hecho un buen espectáculo:

For example, let's see what happens when we accidentally indent the line that thanked the magicians as a group for putting on a good show:

magos.py

magicians.py

```
magicians = ['alice', 'david', 'carolina']
for magician in magicians:
    print(f'{magician.title()}, that was a great trick!')
    print(f'I can't wait to see your next trick, {magician.title()}.\n")
```


❶ `print("Thank you everyone, that was a great magic show!")`

Como la última línea ❶ está sangrada, se imprime una vez por cada persona de la lista:

Because the last line ❶ is indented, it's printed once for each person in the list:

Alice, that was a great trick!
I can't wait to see your next trick, Alice.

Thank you everyone, that was a great magic show!
David, that was a great trick!
I can't wait to see your next trick, David.

Thank you everyone, that was a great magic show!
Carolina, that was a great trick!
I can't wait to see your next trick, Carolina.

Thank you everyone, that was a great magic show!

Este es otro error lógico, similar al de " Olvidar sangrar líneas adicionales " en la página 54 . Como Python no sabe lo que intentas conseguir con tu código, ejecutará todo el código que esté escrito con una sintaxis válida. Si una acción se repite muchas veces cuando sólo debería ejecutarse una vez, probablemente necesites eliminar la sangría del código de esa acción.

This is another logical error, similar to the one in " Forgetting to Indent Additional Lines " on page 54 . Because Python doesn't know what you're trying to accomplish with your code, it will run all code that is written in valid syntax. If an action is repeated many times when it should be executed only once, you probably need to unindent the code for that action.

Olvidar los dos puntos (*Forgetting the Colon*)

Los dos puntos al final de una sentencia `for` indican a Python que interprete la línea siguiente como el inicio de un bucle.

The colon at the end of a `for` statement tells Python to interpret the next line as the start of a loop.

```
magicians = ['alice', 'david', 'carolina']
```

```
❶ for magician in magicians  
    print(magician)
```

Si accidentalmente olvidas los dos puntos ❶ , obtendrás un error de sintaxis porque Python no sabe exactamente lo que estás intentando hacer:

If you accidentally forget the colon ❶ , you'll get a syntax error because Python doesn't know exactly what you're trying to do:

```
File "magicians.py", line 2  
    for magician in magicians
```

^

SyntaxError: expected ':'

Python no sabe si simplemente olvidaste los dos puntos, o si pretendías escribir código adicional para configurar un bucle más complejo. Si el intérprete puede identificar una posible solución, te sugerirá una, como añadir dos puntos al final de la línea, como hace aquí con la respuesta `expected ':'` . Algunos errores tienen soluciones fáciles y obvias, gracias a las sugerencias de los rastreadores de Python. Algunos errores son mucho más difíciles de resolver, incluso cuando el arreglo final sólo implica un único carácter. No te sientas mal si tardas mucho tiempo en encontrar una pequeña solución; no estás solo en esta experiencia.

Python doesn't know if you simply forgot the colon, or if you meant to write additional code to set up a more complex loop. If the interpreter can identify a possible fix it will suggest one, like adding a colon at the end of a line, as it does here with the response `expected ':'` . Some errors have easy, obvious fixes, thanks to the suggestions in Python's tracebacks. Some errors are much harder to resolve, even when the eventual fix only involves a single character. Don't feel bad when a small fix takes a long time to find; you are absolutely not alone in this experience.

Inténtalo tú mismo (Try It Yourself)

4-1. Pizzas: Piensa en al menos tres tipos de tu pizza favorita. Almacena estos nombres de pizza en una lista, y luego utiliza un bucle `for` para imprimir el nombre de cada pizza.

4-1. Pizzas: Think of at least three kinds of your favorite pizza. Store these pizza names in a list, and then use a `for` loop to print the name of each pizza.

Listado en español:

- Modifica tu bucle `for` para imprimir una frase utilizando el nombre de la pizza, en lugar de imprimir sólo el nombre de la pizza. Para cada pizza, debes tener una línea de salida que contenga una sentencia simple como *I like pepperoni pizza*.
- Añade una línea al final de tu programa, fuera del bucle `for`, que diga cuánto te gusta la pizza. La salida debe consistir en tres o más líneas sobre los tipos de pizza que te gustan y luego una frase adicional, como por ejemplo *I really love pizza!*

Listing in English:

- Modify your `for` loop to print a sentence using the name of the pizza, instead of printing just the name of the pizza. For each pizza, you should have one line of output containing a simple statement like *I like pepperoni pizza*.
- Add a line at the end of your program, outside the `for` loop, that states how much you like pizza. The output should consist of three or more lines about the kinds of pizza you like and then an additional sentence, such as *I really love pizza!*

4-2. Animales: Piensa en al menos tres animales diferentes que tengan una característica común. Almacena los nombres de estos animales en una lista y, a continuación, utiliza un bucle `for` para imprimir el nombre de cada animal.

4-2. Animals: Think of at least three different animals that have a common characteristic. Store the names of these animals in a list, and then use a `for` loop to print out the name of each animal.

Listado en español:

- Modifica tu programa para imprimir una declaración sobre cada animal, como por ejemplo *A dog would make a great pet*.
- Añade una línea al final de tu programa, indicando qué tienen en común estos animales. Podrías imprimir una frase como *Any of these animals would make a great pet!*

Listing in English:

- Modify your program to print a statement about each animal, such as *A dog would make a great pet.*
 - Add a line at the end of your program, stating what these animals have in common. You could print a sentence, such as *Any of these animals would make a great pet!*
-

Hacer listas numéricas (Making Numerical Lists)

Existen muchas razones para almacenar un conjunto de números. Por ejemplo, necesitarás llevar un registro de las posiciones de cada personaje en un juego, y puede que quieras que también lleve un registro de las puntuaciones más altas de un jugador. En las visualizaciones de datos, casi siempre trabajarás con conjuntos de números, como temperaturas, distancias, tamaños de población o valores de latitud y longitud, entre otros tipos de conjuntos numéricos.

Many reasons exist to store a set of numbers. For example, you'll need to keep track of the positions of each character in a game, and you might want to keep track of a player's high scores as well. In data visualizations, you'll almost always work with sets of numbers, such as temperatures, distances, population sizes, or latitude and longitude values, among other types of numerical sets.

Las listas son ideales para almacenar conjuntos de números, y Python proporciona diversas herramientas para ayudarte a trabajar eficazmente con listas de números. Una vez que entiendas cómo utilizar estas herramientas con eficacia, tu código funcionará bien incluso cuando tus listas contengan millones de elementos.

Lists are ideal for storing sets of numbers, and Python provides a variety of tools to help you work efficiently with lists of numbers. Once you understand how to use these tools effectively, your code will work well even when your lists contain millions of items.

Uso de la función range() (Using the range() Function)

La función `range()` de Python facilita la generación de una serie de números. Por ejemplo, puedes utilizar la función `range()` para imprimir una serie de números como ésta:

Python's `range()` function makes it easy to generate a series of numbers. For example, you can use the `range()` function to print a series of numbers like this:

primeros_numeros.py

first_numbers.py

```
for value in range(1, 5):  
    print(value)
```

Aunque parece que este código debería imprimir los números del 1 al 5, no imprime el número 5:

Although this code looks like it should print the numbers from 1 to 5, it doesn't print the number 5:

```
1  
2  
3  
4
```

En este ejemplo, `range()` sólo imprime los números del 1 al 4. Este es otro resultado del comportamiento "de uno en uno" que verás a menudo en los lenguajes de programación. La función `range()` hace que Python empiece a contar en el primer valor que le das, y se detiene cuando alcanza el segundo valor que le proporcionas. Como se detiene en ese segundo valor, la salida nunca contiene el valor final, que en este caso habría sido 5.

In this example, `range()` prints only the numbers 1 through 4. This is another result of the off-by-one behavior you'll see often in programming languages. The `range()` function causes Python to start counting at the first value you give it, and it stops when it reaches the second value you provide. Because it stops at that second value, the output never contains the end value, which would have been 5 in this case.

Para imprimir los números del 1 al 5, utilizarías `range(1, 6)` :

To print the numbers from 1 to 5, you would use `range(1, 6)` :

```
for value in range(1, 6):  
    print(value)
```

Esta vez la salida empieza en 1 y termina en 5:

This time the output starts at 1 and ends at 5:

```
1  
2  
3  
4  
5
```

Si tu salida es distinta de la que esperas cuando utilizas `range()`, prueba a ajustar tu valor final en 1.

If your output is different from what you expect when you're using `range()`, try adjusting your end value by 1.

También puedes pasar a `range()` sólo un argumento, y empezará la secuencia de números en 0. Por ejemplo, `range(6)` devolvería los números del 0 al 5.

You can also pass `range()` only one argument, and it will start the sequence of numbers at 0. For example, `range(6)` would return the numbers from 0 through 5.

Utilizar `range()` para hacer una lista de números *(Using `range()` to Make a List of Numbers)*

Si quieres hacer una lista de números, puedes convertir los resultados de `range()` directamente en una lista utilizando la función `list()`. Cuando envuelvas `list()` en una llamada a la función `range()`, la salida será una lista de números.

If you want to make a list of numbers, you can convert the results of `range()` directly into a list using the `list()` function. When you wrap `list()` around a call to the `range()` function, the output will be a list of numbers.

En el ejemplo de la sección anterior, simplemente imprimimos una serie de números. Podemos utilizar `list()` para convertir esa misma serie de números en una lista:

In the example in the previous section, we simply printed out a series of numbers. We can use `list()` to convert that same set of numbers into a list:

```
numbers = list(range(1, 6))  
print(numbers)
```

Este es el resultado:

This is the result:

```
[1, 2, 3, 4, 5]
```

También podemos utilizar la función `range()` para decirle a Python que se salte los números de un rango determinado. Si pasas un tercer argumento a `range()`, Python utiliza ese valor como tamaño de paso al generar los números.

We can also use the `range()` function to tell Python to skip numbers in a given range. If you pass a third argument to `range()`, Python uses that value as a step size when generating numbers.

Por ejemplo, aquí tienes cómo hacer una lista de los números pares entre 1 y 10:

For example, here's how to list the even numbers between 1 and 10:

números_pares.py

even_numbers.py

```
even_numbers = list(range(2, 11, 2))  
print(even_numbers)
```

En este ejemplo, la función `range()` comienza con el valor 2 y luego añade 2 a ese valor. Suma 2 repetidamente hasta que alcanza o supera el valor final, 11, y produce este resultado:

In this example, the `range()` function starts with the value 2 and then adds 2 to that value. It adds 2 repeatedly until it reaches or passes the end value, 11, and produces this result:

```
[2, 4, 6, 8, 10]
```

Puedes crear casi cualquier conjunto de números que desees utilizando la función `range()`. Por ejemplo, considera cómo podrías hacer una lista de los 10 primeros números cuadrados (es decir, el cuadrado de cada número entero del 1 al 10). En Python, dos asteriscos (`**`) representan exponentes. He aquí cómo podrías poner los 10 primeros números cuadrados en una lista:

You can create almost any set of numbers you want to using the `range()` function. For example, consider how you might make a list of the first 10 square numbers (that is, the square of each integer from 1 through 10). In Python, two asterisks (`**`) represent exponents. Here's how you might put the first 10 square numbers into a list:

números_cuadrados.py

square_numbers.py

```
squares = []  
for value in range(1, 11):
```


- ❶ `square = value ** 2`
- ❷ `squares.append(square)`

```
print(squares)
```

Empezamos con una lista vacía llamada `squares` . A continuación, le decimos a Python que repase cada valor del 1 al 10 utilizando la función `range()` . Dentro del bucle, el valor actual se eleva a la segunda potencia y se asigna a la variable `square` ❶ . A continuación, cada nuevo valor de `square` se añade a la lista `squares` ❷ . Por último, cuando el bucle ha terminado de ejecutarse, se imprime la lista de cuadrados:

We start with an empty list called `squares` . Then, we tell Python to loop through each value from 1 to 10 using the `range()` function. Inside the loop, the current value is raised to the second power and assigned to the variable `square` ❶ . Each new value of `square` is then appended to the list `squares` ❷ . Finally, when the loop has finished running, the list of squares is printed:

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Para escribir este código de forma más concisa, omite la variable temporal `square` y añade cada nuevo valor directamente a la lista:

To write this code more concisely, omit the temporary variable `square` and append each new value directly to the list:

```
squares = []  
for value in range(1,11):  
    squares.append(value**2)
```

```
print(squares)
```

Esta línea hace el mismo trabajo que las líneas dentro del bucle `for` del listado anterior. Cada valor del bucle se eleva a la segunda potencia y se añade inmediatamente a la lista de cuadrados.

This line does the same work as the lines inside the `for` loop in the previous listing. Each value in the loop is raised to the second power and then immediately appended to the list of squares.

Puedes utilizar cualquiera de estos métodos cuando hagas listas más complejas. A veces, utilizar una variable temporal hace que tu código sea más fácil de leer; otras veces hace que el código sea innecesariamente largo. Céntrate primero en escribir código que entiendas claramente y que haga lo que quieres que haga. Luego busca enfoques más eficientes a medida que revises tu código.

You can use either of these approaches when you're making more complex lists. Sometimes using a temporary variable makes your code easier to read; other times it makes the code unnecessarily long. Focus first on writing code that you understand clearly, and does what you want it to do. Then look for more efficient approaches as you review your code.

Estadísticas sencillas con una lista de números *(Simple Statistics with a List of Numbers)*

Algunas funciones de Python son útiles cuando se trabaja con listas de números. Por ejemplo, puedes encontrar fácilmente el mínimo, el máximo y la suma de una lista de números:

A few Python functions are helpful when working with lists of numbers. For example, you can easily find the minimum, maximum, and sum of a list of numbers:

```
>>> digits = [1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
>>> min(digits)
0
>>> max(digits)
9
>>> sum(digits)
45
```

Nota *(Note)*

Los ejemplos de esta sección utilizan listas cortas de números que caben fácilmente en la página. Funcionarían igual de bien si tu lista contuviera un millón de números o más.

The examples in this section use short lists of numbers that fit easily on the page. They would work just as well if your list contained a million or more numbers.

Comprensión de listas (*List Comprehensions*)

El método descrito anteriormente para generar la lista `squares` consistía en utilizar tres o cuatro líneas de código. Un *list comprehension* te permite generar esta misma lista en una sola línea de código. Una comprensión de lista combina el bucle `for` y la creación de nuevos elementos en una sola línea, y añade automáticamente cada nuevo elemento. Las comprensiones de lista no siempre se presentan a los principiantes, pero las he incluido aquí porque lo más probable es que las veas en cuanto empieces a mirar el código de otras personas.

The approach described earlier for generating the list `squares` consisted of using three or four lines of code. A *list comprehension* allows you to generate this same list in just one line of code. A list comprehension combines the `for` loop and the creation of new elements into one line, and automatically appends each new element. List comprehensions are not always presented to beginners, but I've included them here because you'll most likely see them as soon as you start looking at other people's code.

El siguiente ejemplo construye la misma lista de números cuadrados que has visto antes, pero utiliza una comprensión de lista:

The following example builds the same list of square numbers you saw earlier but uses a list comprehension:

cuadrados.py

squares.py

```
squares = [value**2 for value in range(1, 11)]  
print(squares)
```

Para utilizar esta sintaxis, empieza con un nombre descriptivo para la lista, como `squares`. A continuación, abre un conjunto de corchetes y define la expresión para los valores que quieres almacenar en la nueva lista. En este ejemplo, la expresión es `value**2`, que eleva

el valor a la segunda potencia. A continuación, escribe un bucle `for` para generar los números que quieras introducir en la expresión, y cierra los corchetes. El bucle `for` de este ejemplo es `for value in range(1, 11)` , que introduce los valores del 1 al 10 en la expresión `value**2` . Observa que no se utilizan dos puntos al final de la expresión `for` .

To use this syntax, begin with a descriptive name for the list, such as `squares` . Next, open a set of square brackets and define the expression for the values you want to store in the new list. In this example the expression is `value**2` , which raises the value to the second power. Then, write a `for` loop to generate the numbers you want to feed into the expression, and close the square brackets. The `for` loop in this example is `for value in range(1, 11)` , which feeds the values 1 through 10 into the expression `value**2` . Note that no colon is used at the end of the `for` statement.

El resultado es la misma lista de números cuadrados que has visto antes:

The result is the same list of square numbers you saw earlier:

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Escribir tus propias comprensiones de listas requiere práctica, pero te merecerá la pena una vez que te sientas cómodo creando listas normales. Cuando estés escribiendo tres o cuatro líneas de código para generar listas y empiece a parecerte repetitivo, plantéate escribir tus propias comprensiones de listas.

It takes practice to write your own list comprehensions, but you'll find them worthwhile once you become comfortable creating ordinary lists. When you're writing three or four lines of code to generate lists and it begins to feel repetitive, consider writing your own list comprehensions.

Pruébalo tú mismo *(Try It Yourself)*

4-3. Contar hasta veinte: Utiliza un bucle `for` para imprimir los números del 1 al 20, ambos inclusive.

4-3. Counting to Twenty: Use a `for` loop to print the numbers from 1 to 20, inclusive.

4-4. Un millón: Haz una lista de los números del uno al millón, y luego utiliza un bucle `for` para imprimir los números. (Si la salida tarda demasiado, deténla pulsando CTRL-C o cerrando la ventana de salida).

4-4. One Million: Make a list of the numbers from one to one million, and then use a `for` loop to print the numbers. (If the output is taking too long, stop it by pressing CTRL-C or by closing the output window.)

4-5. Suma de un millón: Haz una lista de los números del uno al millón, y luego utiliza `min()` y `max()` para asegurarte de que tu lista empieza realmente en uno y termina en un millón. Además, utiliza la función `sum()` para ver lo rápido que Python puede sumar un millón de números.

4-5. Summing a Million: Make a list of the numbers from one to one million, and then use `min()` and `max()` to make sure your list actually starts at one and ends at one million. Also, use the `sum()` function to see how quickly Python can add a million numbers.

4-6. Números impares: Utiliza el tercer argumento de la función `range()` para hacer una lista de los números impares del 1 al 20. Utiliza un bucle `for` para imprimir cada número.

4-6. Odd Numbers: Use the third argument of the `range()` function to make a list of the odd numbers from 1 to 20. Use a `for` loop to print each number.

4-7. Treses: Haz una lista de los múltiplos de 3, del 3 al 30. Utiliza un bucle `for` para imprimir los números de tu lista.

4-7. Threes: Make a list of the multiples of 3, from 3 to 30. Use a `for` loop to print the numbers in your list.

4-8. Cubos: Un número elevado a la tercera potencia se llama *cube* . Por ejemplo, el cubo de 2 se escribe como `2**3` en Python. Haz una lista de los 10 primeros cubos (es decir, el cubo de cada número entero del 1 al 10), y utiliza un bucle `for` para imprimir el valor de cada cubo.

4-8. Cubes: A number raised to the third power is called a *cube* . For example, the cube of 2 is written as `2**3` in Python. Make a list of the first 10 cubes (that is, the cube of each integer from 1 through 10), and use a `for` loop to print out the value of each cube.

4-9. Comprensión de cubos: Utiliza una comprensión de lista para generar una lista de los 10 primeros cubos.

4-9. Cube Comprehension: Use a list comprehension to generate a list of the first 10 cubes.

Trabajar con parte de una lista (*Working with Part of a List*)

En el Capítulo 3 aprendiste a acceder a elementos individuales de una lista, y en este capítulo has aprendido a trabajar con todos los elementos de una lista. También puedes trabajar con un grupo específico de elementos de una lista, que en Python se llama *slice* .

In Chapter 3 you learned how to access single elements in a list, and in this chapter you've been learning how to work through all the elements in a list. You can also work with a specific group of items in a list, called a *slice* in Python.

Cortar una lista (*Slicing a List*)

Para hacer un corte, especifica el índice del primer y último elemento con el que quieres trabajar. Al igual que con la función `range()` , Python se detiene un elemento antes del segundo índice que especifiques. Para obtener los tres primeros elementos de una lista, solicitarías los índices `0` a `3` , lo que devolvería los elementos `0` , `1` y `2` .

To make a slice, you specify the index of the first and last elements you want to work with. As with the `range()` function, Python stops one item before the second index you specify.

To output the first three elements in a list, you would request indices 0 through 3 , which would return elements 0 , 1 , and 2 .

El siguiente ejemplo se refiere a una lista de jugadores de un equipo:

The following example involves a list of players on a team:

jugadores.py

players.py

```
players = ['charles', 'martina', 'michael', 'florence', 'eli']  
print(players[0:3])
```

Este código imprime una porción de la lista. La salida conserva la estructura de la lista, e incluye los tres primeros jugadores de la lista:

This code prints a slice of the list. The output retains the structure of the list, and includes the first three players in the list:

```
['charles', 'martina', 'michael']
```

Puedes generar cualquier subconjunto de una lista. Por ejemplo, si quieres los elementos segundo, tercero y cuarto de una lista, empezarías el trozo en el índice 1 y lo terminarías en el índice 4 :

You can generate any subset of a list. For example, if you want the second, third, and fourth items in a list, you would start the slice at index 1 and end it at index 4 :

```
players = ['charles', 'martina', 'michael', 'florence', 'eli']  
print(players[1:4])
```

Esta vez el corte comienza en 'martina' y termina en 'florence' :

This time the slice starts with 'martina' and ends with 'florence' :

```
['martina', 'michael', 'florence']
```

Si omites el primer índice en un trozo, Python inicia automáticamente tu trozo al principio de la lista:

If you omit the first index in a slice, Python automatically starts your slice at the beginning of the list:

```
players = ['charles', 'martina', 'michael', 'florence', 'eli']  
print(players[:4])
```

Sin índice inicial, Python comienza al principio de la lista:

Without a starting index, Python starts at the beginning of the list:

```
['charles', 'martina', 'michael', 'florence']
```

Una sintaxis similar funciona si quieres un trozo que incluya el final de una lista. Por ejemplo, si quieres todos los elementos desde el tercero hasta el último, puedes empezar con el índice 2 y omitir el segundo índice:

A similar syntax works if you want a slice that includes the end of a list. For example, if you want all items from the third item through the last item, you can start with index 2 and omit the second index:

```
players = ['charles', 'martina', 'michael', 'florence', 'eli']  
print(players[2:])
```


Python devuelve todos los elementos desde el tercer elemento hasta el final de la lista:

Python returns all items from the third item through the end of the list:

```
['michael', 'florence', 'eli']
```

Esta sintaxis te permite obtener todos los elementos desde cualquier punto de tu lista hasta el final, independientemente de la longitud de la lista. Recuerda que un índice negativo devuelve un elemento a cierta distancia del final de una lista; por tanto, puedes dar salida a cualquier trozo del final de una lista. Por ejemplo, si queremos obtener los nombres de los tres últimos jugadores de la lista, podemos utilizar el corte `players[-3:]` :

This syntax allows you to output all of the elements from any point in your list to the end, regardless of the length of the list. Recall that a negative index returns an element a certain distance from the end of a list; therefore, you can output any slice from the end of a list. For example, if we want to output the last three players on the roster, we can use the slice `players[-3:]` :

```
players = ['charles', 'martina', 'michael', 'florence', 'eli']  
print(players[-3:])
```

Esto imprime los nombres de los tres últimos jugadores y seguirá funcionando a medida que la lista de jugadores cambie de tamaño.

This prints the names of the last three players and will continue to work as the list of players changes in size.

Nota (Note)

Puedes incluir un tercer valor entre los corchetes indicando un corte. Si se incluye un tercer valor, esto indica a Python cuántos elementos debe omitir entre los elementos del rango especificado.

You can include a third value in the brackets indicating a slice. If a third value is included, this tells Python how many items to skip between items in the specified range.

Recorrer una porción (*Looping Through a Slice*)

Puedes utilizar una porción en un bucle `for` si quieres recorrer un subconjunto de elementos de una lista. En el siguiente ejemplo, recorreremos los tres primeros jugadores e imprimimos sus nombres como parte de una lista simple:

You can use a slice in a `for` loop if you want to loop through a subset of the elements in a list. In the next example, we loop through the first three players and print their names as part of a simple roster:

```
players = ['charles', 'martina', 'michael', 'florence', 'eli']
```

```
print("Here are the first three players on my team:")
```

```
❶ for player in players[:3]:  
    print(player.title())
```

En lugar de recorrer toda la lista de jugadores, Python recorre sólo los tres primeros nombres ❶ :

Instead of looping through the entire list of players, Python loops through only the first three names ❶ :

Here are the first three players on my team:

Charles

Martina

Michael

Los trozos son muy útiles en varias situaciones. Por ejemplo, cuando estás creando un juego, podrías añadir la puntuación final de un jugador a una lista cada vez que ese jugador termine de jugar. Luego podrías obtener las tres mejores puntuaciones de un jugador ordenando la lista en orden decreciente y tomando una porción que incluya sólo las tres primeras puntuaciones. Cuando trabajes con datos, puedes utilizar trozos para procesarlos en partes de un tamaño determinado. O, cuando estás construyendo una aplicación web, puedes utilizar trozos para mostrar información en una serie de páginas con una cantidad

adecuada de información en cada página.

Slices are very useful in a number of situations. For instance, when you're creating a game, you could add a player's final score to a list every time that player finishes playing. You could then get a player's top three scores by sorting the list in decreasing order and taking a slice that includes just the first three scores. When you're working with data, you can use slices to process your data in chunks of a specific size. Or, when you're building a web application, you could use slices to display information in a series of pages with an appropriate amount of information on each page.

Copiar una lista *(Copying a List)*

A menudo, querrás empezar con una lista existente y hacer una lista completamente nueva basada en la primera. Exploremos cómo funciona copiar una lista y examinemos una situación en la que copiar una lista resulta útil.

Often, you'll want to start with an existing list and make an entirely new list based on the first one. Let's explore how copying a list works and examine one situation in which copying a list is useful.

Para copiar una lista, puedes hacer un corte que incluya toda la lista original omitiendo el primer índice y el segundo índice (`[:]`). Esto le dice a Python que haga un corte que empiece en el primer elemento y termine en el último, produciendo una copia de toda la lista.

To copy a list, you can make a slice that includes the entire original list by omitting the first index and the second index (`[:]`). This tells Python to make a slice that starts at the first item and ends with the last item, producing a copy of the entire list.

Por ejemplo, imagina que tenemos una lista de nuestras comidas favoritas y queremos hacer otra lista de las comidas que le gustan a un amigo. A este amigo le gusta todo lo que hay en nuestra lista hasta ahora, así que podemos crear su lista copiando la nuestra:

For example, imagine we have a list of our favorite foods and want to make a separate list of foods that a friend likes. This friend likes everything in our list so far, so we can create their list by copying ours:

alimentos.py

foods.py

```
my_foods = ['pizza', 'falafel', 'carrot cake']  
❶ friend_foods = my_foods[:]
```

```
print("My favorite foods are:")  
print(my_foods)
```

```
print("\nMy friend's favorite foods are:")  
print(friend_foods)
```

Primero, hacemos una lista de los alimentos que nos gustan llamada `my_foods` . Luego hacemos una nueva lista llamada `friend_foods` . Hacemos una copia de `my_foods` pidiendo un trozo de `my_foods` sin especificar ningún índice ❶ , y asignamos la copia a `friend_foods` . Cuando imprimimos cada lista, vemos que ambas contienen los mismos alimentos:

First, we make a list of the foods we like called `my_foods` . Then we make a new list called `friend_foods` . We make a copy of `my_foods` by asking for a slice of `my_foods` without specifying any indices ❶ , and assign the copy to `friend_foods` . When we print each list, we see that they both contain the same foods:

```
My favorite foods are:  
['pizza', 'falafel', 'carrot cake']
```

```
My friend's favorite foods are:  
['pizza', 'falafel', 'carrot cake']
```

Para demostrar que en realidad tenemos dos listas distintas, añadiremos un nuevo alimento a cada lista y demostraremos que cada lista lleva la cuenta de los alimentos favoritos de la persona correspondiente:

To prove that we actually have two separate lists, we'll add a new food to each list and show that each list keeps track of the appropriate person's favorite foods:

```
my_foods = ['pizza', 'falafel', 'carrot cake']
```

```
❶ friend_foods = my_foods[:]
```

```
❷ my_foods.append('cannoli')
```

```
❸ friend_foods.append('ice cream')
```

```
print("My favorite foods are:")
```

```
print(my_foods)
```

```
print("\nMy friend's favorite foods are:")
```

```
print(friend_foods)
```

Copiamos los elementos originales de `my_foods` en la nueva lista `friend_foods`, como hicimos en el ejemplo anterior ❶. A continuación, añadimos un nuevo alimento a cada lista: añadimos 'cannoli' a `my_foods` ❷, y añadimos 'ice cream' a `friend_foods` ❸. A continuación, imprimimos las dos listas para ver si cada uno de estos alimentos está en la lista adecuada:

We copy the original items in `my_foods` to the new list `friend_foods`, as we did in the previous example ❶. Next, we add a new food to each list: we add 'cannoli' to `my_foods` ❷, and we add 'ice cream' to `friend_foods` ❸. We then print the two lists to see whether each of these foods is in the appropriate list:

```
My favorite foods are:
```

```
['pizza', 'falafel', 'carrot cake', 'cannoli']
```

```
My friend's favorite foods are:
```

```
['pizza', 'falafel', 'carrot cake', 'ice cream']
```

La salida muestra que 'cannoli' aparece ahora en nuestra lista de alimentos favoritos, pero 'ice cream' no. Podemos ver que 'ice cream' aparece ahora en la lista de nuestro amigo pero 'cannoli' no. Si simplemente hubiéramos establecido `friend_foods` igual a `my_foods`, no produciríamos dos listas separadas. Por ejemplo, esto es lo que ocurre cuando intentas copiar una lista sin utilizar un corte:

The output shows that 'cannoli' now appears in our list of favorite foods but 'ice cream' does not. We can see that 'ice cream' now appears in our friend's list but 'cannoli' does not. If we had simply set `friend_foods` equal to `my_foods`, we would not produce two separate lists. For example, here's what happens when you try to copy a list without using a slice:

```
my_foods = ['pizza', 'falafel', 'carrot cake']
```

```
# This doesn't work:
```

```
friend_foods = my_foods
```

```
my_foods.append('cannoli')
```

```
friend_foods.append('ice cream')
```

```
print("My favorite foods are:")
```

```
print(my_foods)
```

```
print("\nMy friend's favorite foods are:")
```

```
print(friend_foods)
```

En lugar de asignar una copia de `my_foods` a `friend_foods`, establecemos `friend_foods` igual a `my_foods`. Esta sintaxis en realidad le dice a Python que asocie la nueva variable `friend_foods` a la lista que ya está asociada a `my_foods`, de modo que ahora ambas variables apuntan a la misma lista. Como resultado, cuando añadamos 'cannoli' a `my_foods`, también aparecerá en `friend_foods`. Del mismo modo, 'ice cream' aparecerá en ambas listas, aunque parezca que sólo se ha añadido a `friend_foods`.

Instead of assigning a copy of `my_foods` to `friend_foods`, we set `friend_foods` equal to `my_foods`. This syntax actually tells Python to associate the new variable `friend_foods` with the list that is already associated with `my_foods`, so now both variables point to the same list. As a result, when we add 'cannoli' to `my_foods`, it will also appear in `friend_foods`. Likewise 'ice cream' will appear in both lists, even though it appears to be added only to `friend_foods`.

La salida muestra que ahora ambas listas son iguales, que no es lo que queríamos:

The output shows that both lists are the same now, which is not what we wanted:

My favorite foods are:

```
['pizza', 'falafel', 'carrot cake', 'cannoli', 'ice cream']
```

My friend's favorite foods are:

```
['pizza', 'falafel', 'carrot cake', 'cannoli', 'ice cream']
```

Nota (Note)

No te preocupes por los detalles de este ejemplo por ahora. Si intentas trabajar con una copia de una lista y observas un comportamiento inesperado, asegúrate de que estás copiando la lista utilizando un corte, como hicimos en el primer ejemplo.

Don't worry about the details in this example for now. If you're trying to work with a copy of a list and you see unexpected behavior, make sure you are copying the list using a slice, as we did in the first example.

Pruébalo tú mismo (Try It Yourself)

4-10. Rebanadas: Utilizando uno de los programas que has escrito en este capítulo, añade varias líneas al final del programa que hagan lo siguiente:

4-10. Slices: Using one of the programs you wrote in this chapter, add several lines to the end of the program that do the following:

Listado en español:

- Imprimir el mensaje *The first three items in the list are:* . A continuación, utiliza una rebanada para imprimir los tres primeros elementos de la lista de ese programa.
- Imprime el mensaje *Three items from the middle of the list are:* . A continuación, utiliza un corte para imprimir tres elementos de la mitad de la lista.
- Imprime el mensaje *The last three items in the list are:* . A continuación, utiliza un corte para imprimir los tres últimos elementos de la lista.

Listing in English:

- Print the message *The first three items in the list are:* . Then use a slice to print the first three items from that program's list.
- Print the message *Three items from the middle of the list are:* . Then use a slice to print three items from the middle of the list.
- Print the message *The last three items in the list are:* . Then use a slice to print the last three items in the list.

4-11. Mis Pizzas, Tus Pizzas: Empieza con tu programa del Ejercicio 4-1 (página 56). Haz una copia de la lista de pizzas y llámala `friend_pizzas` . Luego, haz lo siguiente:

4-11. My Pizzas, Your Pizzas: Start with your program from Exercise 4-1 (page 56). Make a copy of the list of pizzas, and call it `friend_pizzas` . Then, do the following:

Listado en español:

- Añade una nueva pizza a la lista original.
- Añade otra pizza a la lista `friend_pizzas` .
- Demuestra que tienes dos listas separadas. Imprime el mensaje *My favorite pizzas are:* , y luego utiliza un bucle `for` para imprimir la primera lista. Imprime el mensaje *My friend's favorite pizzas are:* , y luego utiliza un bucle `for` para imprimir la segunda lista. Asegúrate de que cada nueva pizza se almacena en la lista adecuada.

Listing in English:

- Add a new pizza to the original list.
- Add a different pizza to the list `friend_pizzas` .
- Prove that you have two separate lists. Print the message *My favorite pizzas are:* , and then use a `for` loop to print the first list. Print the message *My friend's favorite pizzas are:* , and then use a `for` loop to print the second list. Make sure each new pizza is stored in the appropriate list.

4-12. Más bucles: Todas las versiones de *foods.py* de esta sección han evitado utilizar bucles `for` al imprimir, para ahorrar espacio. Elige una versión de *foods.py* , y escribe dos bucles `for` para imprimir cada lista de alimentos.

4-12. More Loops: All versions of *foods.py* in this section have avoided using `for` loops when printing, to save space. Choose a version of *foods.py* , and write two `for` loops to print each list of foods.

Tuplas (Tuples)

Las listas funcionan bien para almacenar colecciones de elementos que pueden cambiar a lo largo de la vida de un programa. La capacidad de modificar listas es especialmente importante cuando trabajas con una lista de usuarios en un sitio web o una lista de personajes en un juego. Sin embargo, a veces querrás crear una lista de elementos que no puedan cambiar. Las tuplas te permiten hacer precisamente eso. Python se refiere a los valores que no pueden cambiar como *immutable*, y una lista inmutable se llama *tuple*.

Lists work well for storing collections of items that can change throughout the life of a program. The ability to modify lists is particularly important when you're working with a list of users on a website or a list of characters in a game. However, sometimes you'll want to create a list of items that cannot change. Tuples allow you to do just that. Python refers to values that cannot change as *immutable*, and an immutable list is called a *tuple*.

Definir una tupla (Defining a Tuple)

Una tupla tiene el mismo aspecto que una lista, salvo que utilizas paréntesis en lugar de corchetes. Una vez definida una tupla, puedes acceder a los elementos individuales utilizando el índice de cada elemento, igual que harías con una lista.

A tuple looks just like a list, except you use parentheses instead of square brackets. Once you define a tuple, you can access individual elements by using each item's index, just as you would for a list.

Por ejemplo, si tenemos un rectángulo que siempre debe tener un tamaño determinado, podemos asegurarnos de que su tamaño no cambie poniendo las dimensiones en una tupla:

For example, if we have a rectangle that should always be a certain size, we can ensure that its size doesn't change by putting the dimensions into a tuple:

dimensiones.py

dimensions.py

```
dimensions = (200, 50)
print(dimensions[0])
print(dimensions[1])
```

Definimos la tupla `dimensions` , utilizando paréntesis en lugar de corchetes. A continuación, imprimimos cada elemento de la tupla individualmente, utilizando la misma sintaxis que hemos estado utilizando para acceder a los elementos de una lista:

We define the tuple `dimensions` , using parentheses instead of square brackets. Then we print each element in the tuple individually, using the same syntax we've been using to access elements in a list:

```
200
50
```

Veamos qué ocurre si intentamos cambiar uno de los elementos de la tupla `dimensions` :

Let's see what happens if we try to change one of the items in the tuple `dimensions` :

```
dimensions = (200, 50)
dimensions[0] = 250
```

Este código intenta cambiar el valor de la primera dimensión, pero Python devuelve un error de tipo. Como estamos intentando modificar una tupla, cosa que no se puede hacer con ese tipo de objeto, Python nos dice que no podemos asignar un nuevo valor a un elemento de una tupla:

This code tries to change the value of the first dimension, but Python returns a type error. Because we're trying to alter a tuple, which can't be done to that type of object, Python tells us we can't assign a new value to an item in a tuple:

Traceback (most recent call last):

File "dimensions.py", line 2, in <module>

dimensions[0] = 250

TypeError: 'tuple' object does not support item assignment

Esto es beneficioso porque queremos que Python lance un error cuando una línea de código intente cambiar las dimensiones del rectángulo.

This is beneficial because we want Python to raise an error when a line of code tries to change the dimensions of the rectangle.

Nota (Note)

Las tuplas se definen técnicamente por la presencia de una coma; los paréntesis les dan un aspecto más ordenado y legible. Si quieres definir una tupla con un solo elemento, debes incluir una coma al final:

Tuples are technically defined by the presence of a comma; the parentheses make them look neater and more readable. If you want to define a tuple with one element, you need to include a trailing comma:

```
my_t = (3,)
```

No suele tener sentido construir una tupla con un solo elemento, pero puede ocurrir cuando las tuplas se generan automáticamente.

It doesn't often make sense to build a tuple with one element, but this can happen when tuples are generated automatically.

Recorrer todos los valores de una tupla (*Looping Through All Values in a Tuple*)

Puedes recorrer todos los valores de una tupla utilizando un bucle `for`, igual que hacías con una lista:

You can loop over all the values in a tuple using a `for` loop, just as you did with a list:

```
dimensions = (200, 50)
for dimension in dimensions:
    print(dimension)
```

Python devuelve todos los elementos de la tupla, igual que lo haría con una lista:

Python returns all the elements in the tuple, just as it would for a list:

```
200
50
```

Escribir sobre una tupla (*Writing Over a Tuple*)

Aunque no puedes modificar una tupla, puedes asignar un nuevo valor a una variable que represente una tupla. Por ejemplo, si quisiéramos cambiar las dimensiones de este rectángulo, podríamos redefinir toda la tupla:

Although you can't modify a tuple, you can assign a new value to a variable that represents a tuple. For example, if we wanted to change the dimensions of this rectangle, we could redefine the entire tuple:

```
dimensions = (200, 50)
print("Original dimensions:")
for dimension in dimensions:
    print(dimension)

dimensions = (400, 100)
print("\nModified dimensions:")
for dimension in dimensions:
    print(dimension)
```

Las cuatro primeras líneas definen la tupla original e imprimen las dimensiones iniciales. A continuación, asociamos una nueva tupla a la variable `dimensions`, e imprimimos los nuevos valores. Esta vez Python no produce ningún error, porque reasignar una variable es válido:

The first four lines define the original tuple and print the initial dimensions. We then associate a new tuple with the variable `dimensions`, and print the new values. Python doesn't raise any errors this time, because reassigning a variable is valid:

Original dimensions:

200

50

Modified dimensions:

400

100

Comparadas con las listas, las tuplas son estructuras de datos sencillas. Utilízalas cuando quieras almacenar un conjunto de valores que no deban modificarse a lo largo de la vida de un programa.

When compared with lists, tuples are simple data structures. Use them when you want to store a set of values that should not be changed throughout the life of a program.

Pruébalo tú mismo *(Try It Yourself)*

4-13. Buffet: Un restaurante tipo buffet sólo ofrece cinco alimentos básicos. Piensa en cinco comidas sencillas y almacénalas en una tupla.

4-13. Buffet: A buffet-style restaurant offers only five basic foods. Think of five simple foods, and store them in a tuple.

Listado en español:

- Utiliza un bucle `for` para imprimir cada uno de los alimentos que ofrece el restaurante.
- Intenta modificar uno de los elementos, y asegúrate de que Python rechaza el cambio.
- El restaurante cambia su menú, sustituyendo dos de los elementos por alimentos diferentes. Añade una línea que reescriba la tupla y, a continuación, utiliza un bucle `for` para imprimir cada uno de los elementos del menú revisado.

Listing in English:

- Use a `for` loop to print each food the restaurant offers.
 - Try to modify one of the items, and make sure that Python rejects the change.
 - The restaurant changes its menu, replacing two of the items with different foods. Add a line that rewrites the tuple, and then use a `for` loop to print each of the items on the revised menu.
-

Dar estilo a tu código (*Styling Your Code*)

Ahora que estás escribiendo programas más largos, es una buena idea aprender a dar estilo a tu código de forma coherente. Tómate el tiempo necesario para que tu código sea lo más fácil de leer posible. Escribir código fácil de leer te ayuda a seguir la pista de lo que hacen tus programas y ayuda también a los demás a entender tu código.

Now that you're writing longer programs, it's a good idea to learn how to style your code consistently. Take the time to make your code as easy as possible to read. Writing easy-to-read code helps you keep track of what your programs are doing and helps others understand your code as well.

Los programadores de Python han acordado una serie de convenciones de estilo para garantizar que el código de todos se estructure más o menos de la misma manera. Una vez que hayas aprendido a escribir código Python limpio, deberías ser capaz de entender la estructura general del código Python de cualquier otra persona, siempre que siga las mismas pautas. Si esperas convertirte en un programador profesional en algún momento, deberías empezar a seguir estas directrices lo antes posible para desarrollar buenos hábitos.

Python programmers have agreed on a number of styling conventions to ensure that everyone's code is structured in roughly the same way. Once you've learned to write clean Python code, you should be able to understand the overall structure of anyone else's Python code, as long as they follow the same guidelines. If you're hoping to become a professional programmer at some point, you should begin following these guidelines as soon as possible to develop good habits.

La Guía de Estilo (*The Style Guide*)

Cuando alguien quiere hacer un cambio en el lenguaje Python, escribe un *Python Enhancement Proposal (PEP)*. Uno de los PEP más antiguos es *PEP 8*, que instruye a los

programadores de Python sobre cómo dar estilo a su código. La PEP 8 es bastante larga, pero gran parte de ella se refiere a estructuras de codificación más complejas que las que has visto hasta ahora.

When someone wants to make a change to the Python language, they write a *Python Enhancement Proposal (PEP)*. One of the oldest PEPs is *PEP 8*, which instructs Python programmers on how to style their code. PEP 8 is fairly lengthy, but much of it relates to more complex coding structures than what you've seen so far.

La guía de estilo de Python se escribió teniendo en cuenta que el código se lee más a menudo de lo que se escribe. Escribirás tu código una vez y luego empezarás a leerlo cuando empieces a depurarlo. Cuando añadas funciones a un programa, pasarás más tiempo leyendo tu código. Cuando compartas tu código con otros programadores, ellos también leerán tu código.

The Python style guide was written with the understanding that code is read more often than it is written. You'll write your code once and then start reading it as you begin debugging. When you add features to a program, you'll spend more time reading your code. When you share your code with other programmers, they'll read your code as well.

Si te dan a elegir entre escribir código que sea más fácil de escribir o código que sea más fácil de leer, los programadores de Python casi siempre te animarán a escribir código que sea más fácil de leer. Las siguientes directrices te ayudarán a escribir código claro desde el principio.

Given the choice between writing code that's easier to write or code that's easier to read, Python programmers will almost always encourage you to write code that's easier to read. The following guidelines will help you write clear code from the start.

Sangría (*Indentation*)

PEP 8 recomienda que utilices cuatro espacios por nivel de sangría. Utilizar cuatro espacios mejora la legibilidad, al tiempo que deja espacio para varios niveles de sangría en cada línea.

PEP 8 recommends that you use four spaces per indentation level. Using four spaces improves readability while leaving room for multiple levels of indentation on each line.

En un documento de procesamiento de textos, la gente suele utilizar tabuladores en lugar de espacios para sangrar. Esto funciona bien para los documentos de procesamiento de textos, pero el intérprete de Python se confunde cuando se mezclan tabuladores con espacios. Todos los editores de texto ofrecen una opción que te permite utilizar la tecla TAB, pero que convierte cada tabulación en un número determinado de espacios. Definitivamente, debes utilizar la tecla TAB, pero asegúrate también de que tu editor está configurado para insertar espacios en lugar de tabuladores en tu documento.

In a word processing document, people often use tabs rather than spaces to indent. This works well for word processing documents, but the Python interpreter gets confused when tabs are mixed with spaces. Every text editor provides a setting that lets you use the TAB key but then converts each tab to a set number of spaces. You should definitely use your TAB key, but also make sure your editor is set to insert spaces rather than tabs into your document.

Mezclar tabuladores y espacios en tu archivo puede causar problemas muy difíciles de diagnosticar. Si crees que tienes una mezcla de tabuladores y espacios, puedes convertir todos los tabuladores de un archivo en espacios en la mayoría de los editores.

Mixing tabs and spaces in your file can cause problems that are very difficult to diagnose. If you think you have a mix of tabs and spaces, you can convert all tabs in a file to spaces in most editors.

Longitud de línea (*Line Length*)

Muchos programadores de Python recomiendan que cada línea tenga menos de 80 caracteres. Históricamente, esta directriz se desarrolló porque en la mayoría de los ordenadores sólo cabían 79 caracteres en una sola línea en una ventana de terminal. Actualmente, la gente puede hacer caber líneas mucho más largas en sus pantallas, pero existen otras razones para adherirse a la longitud de línea estándar de 79 caracteres.

Many Python programmers recommend that each line should be less than 80 characters. Historically, this guideline developed because most computers could fit only 79 characters

on a single line in a terminal window. Currently, people can fit much longer lines on their screens, but other reasons exist to adhere to the 79-character standard line length.

Los programadores profesionales suelen tener varios archivos abiertos en la misma pantalla, y utilizar la longitud de línea estándar les permite ver líneas enteras en dos o tres archivos que están abiertos uno al lado del otro en la pantalla. PEP 8 también recomienda que limites todos tus comentarios a 72 caracteres por línea, porque algunas de las herramientas que generan documentación automática para proyectos grandes añaden caracteres de formato al principio de cada línea comentada.

Professional programmers often have several files open on the same screen, and using the standard line length allows them to see entire lines in two or three files that are open side by side onscreen. PEP 8 also recommends that you limit all of your comments to 72 characters per line, because some of the tools that generate automatic documentation for larger projects add formatting characters at the beginning of each commented line.

Las directrices de PEP 8 sobre la longitud de las líneas no son inamovibles, y algunos equipos prefieren un límite de 99 caracteres. No te preocupes demasiado por la longitud de línea en tu código mientras aprendes, pero ten en cuenta que las personas que trabajan en colaboración casi siempre siguen las directrices de PEP 8. La mayoría de los editores te permiten configurar una señal visual, normalmente una línea vertical en tu pantalla, que te indica dónde están estos límites.

The PEP 8 guidelines for line length are not set in stone, and some teams prefer a 99-character limit. Don't worry too much about line length in your code as you're learning, but be aware that people who are working collaboratively almost always follow the PEP 8 guidelines. Most editors allow you to set up a visual cue, usually a vertical line on your screen, that shows you where these limits are.

Nota (Note)

El Apéndice B te muestra cómo configurar tu editor de texto para que inserte siempre cuatro espacios cada vez que pulses la tecla TAB y te muestre una directriz vertical que te ayude a seguir el límite de 79 caracteres.

Appendix B shows you how to configure your text editor so it always inserts four spaces each time you press the TAB key and shows a vertical guideline to help you follow the 79-character limit.

Líneas en blanco (*Blank Lines*)

Para agrupar partes de tu programa visualmente, utiliza líneas en blanco. Debes utilizar líneas en blanco para organizar tus archivos, pero no lo hagas en exceso. Siguiendo los ejemplos de este libro, conseguirás el equilibrio adecuado. Por ejemplo, si tienes cinco líneas de código que construyen una lista y luego otras tres líneas que hacen algo con esa lista, es apropiado colocar una línea en blanco entre las dos secciones. Sin embargo, no debes colocar tres o cuatro líneas en blanco entre las dos secciones.

To group parts of your program visually, use blank lines. You should use blank lines to organize your files, but don't do so excessively. By following the examples provided in this book, you should strike the right balance. For example, if you have five lines of code that build a list and then another three lines that do something with that list, it's appropriate to place a blank line between the two sections. However, you should not place three or four blank lines between the two sections.

Las líneas en blanco no afectarán a la ejecución de tu código, pero sí a su legibilidad. El intérprete de Python utiliza la sangría horizontal para interpretar el significado de tu código, pero no tiene en cuenta el espaciado vertical.

Blank lines won't affect how your code runs, but they will affect the readability of your code. The Python interpreter uses horizontal indentation to interpret the meaning of your code, but it disregards vertical spacing.

Otras directrices de estilo (*Other Style Guidelines*)

PEP 8 tiene muchas recomendaciones de estilo adicionales, pero la mayoría de las directrices se refieren a programas más complejos que lo que estás escribiendo en este momento. A medida que aprendas estructuras de Python más complejas, compartiré las partes relevantes de las directrices de la PEP 8.

PEP 8 has many additional styling recommendations, but most of the guidelines refer to more complex programs than what you're writing at this point. As you learn more complex Python structures, I'll share the relevant parts of the PEP 8 guidelines.

Pruébalo tú mismo (*Try It Yourself*)

4-14. PEP 8 : Echa un vistazo a la guía de estilo original de PEP 8 en <https://python.org/dev/peps/pep-0008> . No usarás mucho de ella ahora, pero puede ser interesante hojearla.

4-14. PEP 8: Look through the original PEP 8 style guide at <https://python.org/dev/peps/pep-0008> . You won't use much of it now, but it might be interesting to skim through it.

4-15. Revisión del código: Elige tres de los programas que has escrito en este capítulo y modifica cada uno de ellos para que cumpla con PEP 8.

4-15. Code Review: Choose three of the programs you've written in this chapter and modify each one to comply with PEP 8.

Listado en español:

- Utiliza cuatro espacios para cada nivel de sangría. Configura tu editor de texto para que inserte cuatro espacios cada vez que pulses la tecla TAB, si aún no lo has hecho (en el Apéndice B encontrarás instrucciones sobre cómo hacerlo).
- Utiliza menos de 80 caracteres en cada línea, y configura tu editor para que muestre una línea vertical en la posición del carácter 80.
- No utilices excesivamente líneas en blanco en tus archivos de programa.

Listing in English:

- Use four spaces for each indentation level. Set your text editor to insert four spaces every time you press the TAB key, if you haven't already done so (see Appendix B for instructions on how to do this).
- Use less than 80 characters on each line, and set your editor to show a vertical guideline at the 80th character position.
- Don't use blank lines excessively in your program files.

Resumen (Summary)

En este capítulo has aprendido a trabajar eficazmente con los elementos de una lista. Aprendiste a trabajar a través de una lista utilizando un bucle `for`, cómo Python utiliza la sangría para estructurar un programa y cómo evitar algunos errores comunes de sangría. Aprendiste a hacer listas numéricas sencillas, así como algunas operaciones que puedes realizar con listas numéricas. Aprendiste a cortar una lista para trabajar con un subconjunto de elementos y a copiar listas correctamente utilizando un corte. También aprendiste sobre las tuplas, que proporcionan cierto grado de protección a un conjunto de valores que no deben cambiar, y cómo dar estilo a tu código, cada vez más complejo, para que sea fácil de leer.

In this chapter, you learned how to work efficiently with the elements in a list. You learned how to work through a list using a `for` loop, how Python uses indentation to structure a program, and how to avoid some common indentation errors. You learned to make simple numerical lists, as well as a few operations you can perform on numerical lists. You learned how to slice a list to work with a subset of items and how to copy lists properly using a slice. You also learned about tuples, which provide a degree of protection to a set of values that shouldn't change, and how to style your increasingly complex code to make it easy to read.

En el Capítulo 5, aprenderás a responder adecuadamente a diferentes condiciones utilizando las sentencias `if`. Aprenderás a encadenar conjuntos relativamente complejos de pruebas condicionales para responder adecuadamente exactamente al tipo de situación o información que buscas. También aprenderás a utilizar las sentencias `if` mientras recorres una lista en bucle para realizar acciones específicas con los elementos seleccionados de una lista.

In Chapter 5, you'll learn to respond appropriately to different conditions by using `if` statements. You'll learn to string together relatively complex sets of conditional tests to respond appropriately to exactly the kind of situation or information you're looking for. You'll also learn to use `if` statements while looping through a list to take specific actions with selected elements from a list.

Capítulo 5: Sentencias if (Chapter 5: if Statements)

5

Sentencias if (5 if Statements)



DESCRIPCIÓN DE LA
IMAGEN (En español):

La imagen muestra una ilustración en blanco y negro de una serpiente con una expresión sonriente y ojos con gafas de aviador. Esta serpiente lleva envuelta en sus anillos lo que parece ser un motor de reacción de un avión. El fondo es de un color gris uniforme.

Programar suele implicar examinar un conjunto de condiciones y decidir qué acción tomar en función de ellas. La sentencia `if` de Python te permite examinar el estado actual de un programa y responder adecuadamente a ese estado.

Programming often involves examining a set of conditions and deciding which action to take based on those conditions. Python's `if` statement allows you to examine the current state of a program and respond appropriately to that state.

En este capítulo, aprenderás a escribir pruebas condicionales, que te permiten comprobar cualquier condición de interés. Aprenderás a escribir sencillas sentencias `if`, y aprenderás a crear una serie más compleja de sentencias `if` para identificar cuándo se dan las condiciones exactas que deseas. A continuación, aplicarás este concepto a las listas, de modo que podrás escribir un bucle `for` que trate la mayoría de los elementos de una lista de una manera, pero que trate determinados elementos con valores específicos de una manera diferente.

In this chapter, you'll learn to write conditional tests, which allow you to check any condition of interest. You'll learn to write simple `if` statements, and you'll learn how to create a more complex series of `if` statements to identify when the exact conditions you

want are present. You'll then apply this concept to lists, so you'll be able to write a `for` loop that handles most items in a list one way but handles certain items with specific values in a different way.

Un ejemplo sencillo (A Simple Example)

El siguiente ejemplo muestra cómo las pruebas `if` te permiten responder correctamente a situaciones especiales. Imagina que tienes una lista de coches y quieres imprimir el nombre de cada coche. Los nombres de los coches son nombres propios, por lo que los nombres de la mayoría de los coches deberían imprimirse en mayúsculas. Sin embargo, el valor `'bmw'` debe imprimirse en mayúsculas. El siguiente código recorre una lista de nombres de coches y busca el valor `'bmw'`. Siempre que el valor sea `'bmw'`, se imprime en mayúsculas en lugar de en mayúsculas de título:

The following example shows how `if` tests let you respond to special situations correctly. Imagine you have a list of cars and you want to print out the name of each car. Car names are proper names, so the names of most cars should be printed in title case. However, the value `'bmw'` should be printed in all uppercase. The following code loops through a list of car names and looks for the value `'bmw'`. Whenever the value is `'bmw'`, it's printed in uppercase instead of title case:

coches.py

cars.py

```
cars = ['audi', 'bmw', 'subaru', 'toyota']
```

```
for car in cars:
```

```
❶ if car == 'bmw':
    print(car.upper())
else:
    print(car.title())
```

El bucle de este ejemplo comprueba primero si el valor actual de `car` es `'bmw'` ❶. Si lo es, el valor se imprime en mayúsculas. Si el valor de `car` es cualquier otra cosa que `'bmw'`, se imprime en mayúsculas:

The loop in this example first checks if the current value of `car` is `'bmw'` ❶ . If it is, the value is printed in uppercase. If the value of `car` is anything other than `'bmw'` , it's printed in title case:

```
Audi
BMW
Subaru
Toyota
```

Este ejemplo combina varios de los conceptos que aprenderás en este capítulo. Empecemos por ver los tipos de pruebas que puedes utilizar para examinar las condiciones de tu programa.

This example combines a number of the concepts you'll learn about in this chapter. Let's begin by looking at the kinds of tests you can use to examine the conditions in your program.

Pruebas condicionales (*Conditional Tests*)

En el corazón de cada sentencia `if` hay una expresión que puede evaluarse como `True` o `False` y que se denomina *conditional test* . Python utiliza los valores `True` y `False` para decidir si debe ejecutarse el código de una sentencia `if` . Si una prueba condicional se evalúa como `True` , Python ejecuta el código que sigue a la sentencia `if` . Si el resultado de la prueba es `False` , Python ignora el código que sigue a la sentencia `if` .

At the heart of every `if` statement is an expression that can be evaluated as `True` or `False` and is called a *conditional test* . Python uses the values `True` and `False` to decide whether the code in an `if` statement should be executed. If a conditional test evaluates to `True` , Python executes the code following the `if` statement. If the test evaluates to `False` , Python ignores the code following the `if` statement.

Comprobación de igualdad (*Checking for Equality*)

La mayoría de las pruebas condicionales comparan el valor actual de una variable con un valor específico de interés. La prueba condicional más sencilla comprueba si el valor de una variable es igual al valor de interés:

Most conditional tests compare the current value of a variable to a specific value of interest. The simplest conditional test checks whether the value of a variable is equal to the value of interest:

```
>>> car = 'bmw'
>>> car == 'bmw'
True
```

La primera línea establece el valor de `car` en `'bmw'` utilizando un único signo igual, como ya has visto muchas veces. La siguiente línea comprueba si el valor de `car` es `'bmw'` utilizando un signo igual doble (`==`). Este *equality operator* devuelve `True` si los valores a izquierda y derecha del operador coinciden, y `False` si no coinciden. Los valores de este ejemplo coinciden, por lo que Python devuelve `True` .

The first line sets the value of `car` to `'bmw'` using a single equal sign, as you've seen many times already. The next line checks whether the value of `car` is `'bmw'` by using a double equal sign (`==`). This *equality operator* returns `True` if the values on the left and right side of the operator match, and `False` if they don't match. The values in this example match, so Python returns `True` .

Cuando el valor de `car` es distinto de `'bmw'` , esta prueba devuelve `False` :

When the value of `car` is anything other than `'bmw'` , this test returns `False` :

```
>>> car = 'audi'
>>> car == 'bmw'
False
```

Un solo signo igual es realmente una sentencia; podrías leer la primera línea de código aquí como "Establece el valor de `car` igual a `'audi'` ." En cambio, un signo igual doble formula una pregunta: "¿Es el valor de `car` igual a `'bmw'` ?" La mayoría de los lenguajes de programación utilizan los signos igual de esta forma.

A single equal sign is really a statement; you might read the first line of code here as "Set the value of `car` equal to `'audi'` ." On the other hand, a double equal sign asks a question:

“Is the value of `car` equal to `'bmw'` ?” Most programming languages use equal signs in this way.

Ignorar mayúsculas y minúsculas al comprobar la igualdad (*Ignoring Case When Checking for Equality*)

En Python, la comprobación de la igualdad distingue entre mayúsculas y minúsculas. Por ejemplo, dos valores con mayúsculas diferentes no se consideran iguales:

Testing for equality is case sensitive in Python. For example, two values with different capitalization are not considered equal:

```
>>> car = 'Audi'
>>> car == 'audi'
False
```

Si las mayúsculas y minúsculas importan, este comportamiento es ventajoso. Pero si las mayúsculas y minúsculas no importan y sólo quieres comprobar el valor de una variable, puedes convertir el valor de la variable a minúsculas antes de hacer la comparación:

If case matters, this behavior is advantageous. But if case doesn't matter and instead you just want to test the value of a variable, you can convert the variable's value to lowercase before doing the comparison:

```
>>> car = 'Audi'
>>> car.lower() == 'audi'
True
```

Esta prueba devolverá `True` independientemente de cómo esté formateado el valor `'Audi'`, porque ahora la prueba no distingue entre mayúsculas y minúsculas. El método `lower()` no cambia el valor almacenado originalmente en `car`, por lo que puedes hacer este tipo de comparación sin afectar a la variable original:

This test will return `True` no matter how the value `'Audi'` is formatted because the test is now case insensitive. The `lower()` method doesn't change the value that was originally stored in `car`, so you can do this kind of comparison without affecting the original variable:

```
>>> car = 'Audi'
>>> car.lower() == 'audi'
True
>>> car
'Audi'
```

Primero asignamos la cadena en mayúsculas 'Audi' a la variable `car`. A continuación, convertimos el valor de `car` a minúsculas y comparamos el valor en minúsculas con la cadena 'audi'. Las dos cadenas coinciden, por lo que Python devuelve `True`. Podemos ver que el valor almacenado en `car` no se ha visto afectado por el método `lower()`.

We first assign the capitalized string 'Audi' to the variable `car`. Then, we convert the value of `car` to lowercase and compare the lowercase value to the string 'audi'. The two strings match, so Python returns `True`. We can see that the value stored in `car` has not been affected by the `lower()` method.

Los sitios web aplican ciertas reglas a los datos que introducen los usuarios de forma similar a ésta. Por ejemplo, un sitio podría utilizar una prueba condicional como ésta para asegurarse de que cada usuario tiene un nombre de usuario realmente único, y no sólo una variación de las mayúsculas del nombre de usuario de otra persona. Cuando alguien introduce un nuevo nombre de usuario, éste se convierte a minúsculas y se compara con las versiones en minúsculas de todos los nombres de usuario existentes. Durante esta comprobación, un nombre de usuario como 'John' será rechazado si ya se utiliza alguna variación de 'john'.

Websites enforce certain rules for the data that users enter in a manner similar to this. For example, a site might use a conditional test like this to ensure that every user has a truly unique username, not just a variation on the capitalization of another person's username. When someone submits a new username, that new username is converted to lowercase and compared to the lowercase versions of all existing usernames. During this check, a username like 'John' will be rejected if any variation of 'john' is already in use.

Comprobación de la desigualdad (*Checking for Inequality*)

Cuando quieras determinar si dos valores no son iguales, puedes utilizar la sentencia *inequality operator* (`!=`). Utilicemos otra sentencia `if` para examinar cómo utilizar el operador de desigualdad. Almacenaremos un ingrediente de pizza solicitado en una

variable y luego imprimiremos un mensaje si la persona no ha pedido anchoas:

When you want to determine whether two values are not equal, you can use the *inequality operator* (`!=`). Let's use another `if` statement to examine how to use the inequality operator. We'll store a requested pizza topping in a variable and then print a message if the person did not order anchovies:

toppings.py

toppings.py

```
requested_topping = 'mushrooms'
```

```
if requested_topping != 'anchovies':  
    print("Hold the anchovies!")
```

Este código compara el valor de `requested_topping` con el valor `'anchovies'` . Si estos dos valores no coinciden, Python devuelve `True` y ejecuta el código que sigue a la sentencia `if` . Si los dos valores coinciden, Python devuelve `False` y no ejecuta el código que sigue a la sentencia `if` .

This code compares the value of `requested_topping` to the value `'anchovies'` . If these two values do not match, Python returns `True` and executes the code following the `if` statement. If the two values match, Python returns `False` and does not run the code following the `if` statement.

Como el valor de `requested_topping` no es `'anchovies'` , se ejecuta la función `print()` :

Because the value of `requested_topping` is not `'anchovies'` , the `print()` function is executed:

Hold the anchovies!

La mayoría de las expresiones condicionales que escribas comprobarán la igualdad, pero a veces te resultará más eficaz comprobar la desigualdad.

Most of the conditional expressions you write will test for equality, but sometimes you'll find it more efficient to test for inequality.

Comparaciones numéricas (*Numerical Comparisons*)

Comprobar valores numéricos es bastante sencillo. Por ejemplo, el siguiente código comprueba si una persona tiene 18 años:

Testing numerical values is pretty straightforward. For example, the following code checks whether a person is 18 years old:

```
>>> age = 18
>>> age == 18
True
```

También puedes comprobar si dos números no son iguales. Por ejemplo, el siguiente código imprime un mensaje si la respuesta dada no es correcta:

You can also test to see if two numbers are not equal. For example, the following code prints a message if the given answer is not correct:

numero_magico.py

magic_number.py

```
answer = 17
if answer != 42:
    print("That is not the correct answer. Please try again!")
```

La prueba condicional pasa, porque el valor de `answer` (`17`) no es igual a `42` . Como la prueba pasa, se ejecuta el bloque de código sangrado:

The conditional test passes, because the value of `answer` (`17`) is not equal to `42` .
Because the test passes, the indented code block is executed:

That is not the correct answer. Please try again!

También puedes incluir varias comparaciones matemáticas en tus sentencias condicionales, como menor que, menor o igual que, mayor que y mayor o igual que:

You can include various mathematical comparisons in your conditional statements as well, such as less than, less than or equal to, greater than, and greater than or equal to:

```
>>> age = 19
>>> age < 21
True
>>> age <= 21
True
>>> age > 21
False
>>> age >= 21
False
```

Cada comparación matemática puede utilizarse como parte de una sentencia `if` , que puede ayudarte a detectar las condiciones exactas de interés.

Each mathematical comparison can be used as part of an `if` statement, which can help you detect the exact conditions of interest.

Comprobar varias condiciones (*Checking Multiple Conditions*)

Puede que quieras comprobar varias condiciones al mismo tiempo. Por ejemplo, a veces puedes necesitar que dos condiciones sean `True` para realizar una acción. Otras veces, puedes estar satisfecho con que sólo una condición sea `True` . Las palabras clave `and` y `or` pueden ayudarte en estas situaciones.

You may want to check multiple conditions at the same time. For example, sometimes you might need two conditions to be `True` to take an action. Other times, you might be satisfied with just one condition being `True`. The keywords `and` and `or` can help you in these situations.

Utilizar y para comprobar varias condiciones (Using and to Check Multiple Conditions)

Para comprobar si dos condiciones son ambas `True` simultáneamente, utiliza la palabra clave `and` para combinar las dos pruebas condicionales; si cada prueba pasa, la expresión global se evalúa como `True`. Si alguna de las pruebas falla o si ambas fallan, la expresión se evalúa como `False`.

To check whether two conditions are both `True` simultaneously, use the keyword `and` to combine the two conditional tests; if each test passes, the overall expression evaluates to `True`. If either test fails or if both tests fail, the expression evaluates to `False`.

Por ejemplo, puedes comprobar si dos personas tienen más de 21 años utilizando la siguiente prueba:

For example, you can check whether two people are both over 21 by using the following test:

```
>>> age_0 = 22
>>> age_1 = 18
❶ >>> age_0 >= 21 and age_1 >= 21
False
❷ >>> age_1 = 22
>>> age_0 >= 21 and age_1 >= 21
True
```

En primer lugar, definimos dos edades, `age_0` y `age_1`. A continuación, comprobamos si ambas edades son mayores de 21 años ❶. La prueba de la izquierda pasa, pero la de la derecha falla, por lo que la expresión condicional global se evalúa como `False`. A continuación, cambiamos `age_1` por 22 ❷. El valor de `age_1` es ahora mayor que 21, por lo que ambas pruebas individuales pasan, haciendo que la expresión condicional global se evalúe como `True`.

First, we define two ages, `age_0` and `age_1` . Then we check whether both ages are 21 or older ❶ . The test on the left passes, but the test on the right fails, so the overall conditional expression evaluates to `False` . We then change `age_1` to 22 ❷ . The value of `age_1` is now greater than 21, so both individual tests pass, causing the overall conditional expression to evaluate as `True` .

Para mejorar la legibilidad, puedes utilizar paréntesis alrededor de las pruebas individuales, pero no es obligatorio. Si utilizas paréntesis, tu prueba tendría este aspecto:

To improve readability, you can use parentheses around the individual tests, but they are not required. If you use parentheses, your test would look like this:

```
(age_0 >= 21) and (age_1 >= 21)
```

Utilizar o para comprobar varias condiciones (Using or to Check Multiple Conditions)

La palabra clave `or` también te permite comprobar varias condiciones, pero se supera cuando se supera una o las dos pruebas individuales. Una expresión `or` sólo falla cuando fallan las dos pruebas individuales.

The keyword `or` allows you to check multiple conditions as well, but it passes when either or both of the individual tests pass. An `or` expression fails only when both individual tests fail.

Consideremos de nuevo dos edades, pero esta vez buscaremos que sólo una persona tenga más de 21 años:

Let's consider two ages again, but this time we'll look for only one person to be over 21:

```
>>> age_0 = 22
>>> age_1 = 18
❶ >>> age_0 >= 21 or age_1 >= 21
True
❷ >>> age_0 = 18
```

```
>>> age_0 >= 21 or age_1 >= 21
False
```

Volvemos a empezar con dos variables de edad. Como la prueba para `age_0` ❶ pasa, la expresión global se evalúa como `True` . A continuación, bajamos `age_0` a 18. En la prueba final ❷ , ahora ambas pruebas fallan y la expresión global se evalúa como `False` .

We start with two age variables again. Because the test for `age_0` ❶ passes, the overall expression evaluates to `True` . We then lower `age_0` to 18. In the final test ❷ , both tests now fail and the overall expression evaluates to `False` .

Comprobar si un valor está en una lista (*Checking Whether a Value Is in a List*)

A veces es importante comprobar si una lista contiene un determinado valor antes de realizar una acción. Por ejemplo, puede que quieras comprobar si un nuevo nombre de usuario ya existe en una lista de nombres de usuario actuales antes de completar el registro de alguien en un sitio web. En un proyecto cartográfico, puede que quieras comprobar si una ubicación enviada ya existe en una lista de ubicaciones conocidas.

Sometimes it's important to check whether a list contains a certain value before taking an action. For example, you might want to check whether a new username already exists in a list of current usernames before completing someone's registration on a website. In a mapping project, you might want to check whether a submitted location already exists in a list of known locations.

Para averiguar si un valor concreto ya está en una lista, utiliza la palabra clave `in` . Consideremos un código que podrías escribir para una pizzería. Haremos una lista de los ingredientes que un cliente ha pedido para una pizza y luego comprobaremos si determinados ingredientes están en la lista.

To find out whether a particular value is already in a list, use the keyword `in` . Let's consider some code you might write for a pizzeria. We'll make a list of toppings a customer has requested for a pizza and then check whether certain toppings are in the list.

```
>>> requested_toppings = ['mushrooms', 'onions', 'pineapple']
>>> 'mushrooms' in requested_toppings
```


True

```
>>> 'pepperoni' in requested_toppings
```

False

La palabra clave `in` indica a Python que compruebe la existencia de `'mushrooms'` y `'pepperoni'` en la lista `requested_toppings`. Esta técnica es bastante potente porque puedes crear una lista de valores esenciales y luego comprobar fácilmente si el valor que estás comprobando coincide con uno de los valores de la lista.

The keyword `in` tells Python to check for the existence of `'mushrooms'` and `'pepperoni'` in the list `requested_toppings`. This technique is quite powerful because you can create a list of essential values, and then easily check whether the value you're testing matches one of the values in the list.

Comprobar si un valor no está en una lista (*Checking Whether a Value Is Not in a List*)

Otras veces, es importante saber si un valor no aparece en una lista. Puedes utilizar la palabra clave `not` en esta situación. Por ejemplo, considera una lista de usuarios que tienen prohibido comentar en un foro. Puedes comprobar si se ha prohibido a un usuario antes de permitir que esa persona envíe un comentario:

Other times, it's important to know if a value does not appear in a list. You can use the keyword `not` in this situation. For example, consider a list of users who are banned from commenting in a forum. You can check whether a user has been banned before allowing that person to submit a comment:

usuarios_prohibidos.py

banned_users.py

```
banned_users = ['andrew', 'carolina', 'david']  
user = 'marie'
```

```
if user not in banned_users:  
    print(f"{user.title()}, you can post a response if you wish.")
```

La declaración `if` se lee aquí con bastante claridad. Si el valor de `user` no está en la lista `banned_users`, Python devuelve `True` y ejecuta la línea sangrada.

The `if` statement here reads quite clearly. If the value of `user` is not in the list `banned_users`, Python returns `True` and executes the indented line.

El usuario 'marie' no está en la lista `banned_users`, por lo que ve un mensaje invitándole a publicar una respuesta:

The user 'marie' is not in the list `banned_users`, so she sees a message inviting her to post a response:

Marie, you can post a response if you wish.

Expresiones booleanas (*Boolean Expressions*)

A medida que aprendas más sobre programación, oirás en algún momento el término *Boolean expression*. Una expresión booleana no es más que otro nombre para una prueba condicional. Un *Boolean value* es `True` o `False`, igual que el valor de una expresión condicional después de haber sido evaluada.

As you learn more about programming, you'll hear the term *Boolean expression* at some point. A Boolean expression is just another name for a conditional test. A *Boolean value* is either `True` or `False`, just like the value of a conditional expression after it has been evaluated.

Los valores booleanos se utilizan a menudo para controlar ciertas condiciones, como si un juego se está ejecutando o si un usuario puede editar cierto contenido en un sitio web:

Boolean values are often used to keep track of certain conditions, such as whether a game is running or whether a user can edit certain content on a website:

```
game_active = True
can_edit = False
```

Los valores booleanos proporcionan una forma eficaz de realizar un seguimiento del estado de un programa o de una condición concreta que sea importante en tu programa.

Boolean values provide an efficient way to track the state of a program or a particular condition that is important in your program.

Pruébalo tú mismo *(Try It Yourself)*

5-1. Pruebas condicionales: Escribe una serie de pruebas condicionales. Imprime una declaración que describa cada prueba y tu predicción para los resultados de cada una de ellas. Tu código debería tener este aspecto:

5-1. Conditional Tests: Write a series of conditional tests. Print a statement describing each test and your prediction for the results of each test. Your code should look something like this:

```
car = 'subaru'
print("Is car == 'subaru'? I predict True.")
print(car == 'subaru')

print("\nIs car == 'audi'? I predict False.")
print(car == 'audi')
```

Listado en español:

- Fíjate bien en tus resultados y asegúrate de que entiendes por qué cada línea se evalúa como True o False .
- Crea al menos 10 pruebas. Haz que al menos 5 pruebas evalúen a True y otras 5 pruebas evalúen a False .

Listing in English:

- Look closely at your results, and make sure you understand why each line evaluates to True or False .
- Create at least 10 tests. Have at least 5 tests evaluate to True and another 5 tests evaluate to False .

5-2. Más pruebas condicionales: No tienes por qué limitar a 10 el número de pruebas que crees. Si quieres probar más comparaciones, escribe más pruebas y añádelas a

`conditional_tests.py` . Ten al menos un resultado de `True` y otro de `False` para cada una de las siguientes:

5-2. More Conditional Tests: You don't have to limit the number of tests you create to 10. If you want to try more comparisons, write more tests and add them to `conditional_tests.py` . Have at least one `True` and one `False` result for each of the following:

Listado en español:

- Pruebas de igualdad y desigualdad con cadenas
- Pruebas con el método `lower()`
- Pruebas numéricas de igualdad y desigualdad, mayor que y menor que, mayor o igual que y menor o igual que
- Pruebas con las palabras clave `and` y `or`
- Comprueba si un elemento está en una lista
- Comprueba si un elemento no está en una lista

Listing in English:

- Tests for equality and inequality with strings
 - Tests using the `lower()` method
 - Numerical tests involving equality and inequality, greater than and less than, greater than or equal to, and less than or equal to
 - Tests using the `and` keyword and the `or` keyword
 - Test whether an item is in a list
 - Test whether an item is not in a list
-

Sentencias `if` (*if Statements*)

Cuando entiendas las pruebas condicionales, podrás empezar a escribir sentencias `if` . Existen varios tipos de sentencias `if` , y tu elección dependerá del número de condiciones que necesites probar. Has visto varios ejemplos de sentencias `if` en la discusión sobre las pruebas condicionales, pero ahora vamos a profundizar en el tema.

When you understand conditional tests, you can start writing `if` statements. Several different kinds of `if` statements exist, and your choice of which to use depends on the number of conditions you need to test. You saw several examples of `if` statements in the discussion about conditional tests, but now let's dig deeper into the topic.

Sentencias if simples (*Simple if Statements*)

El tipo más sencillo de sentencia `if` tiene una prueba y una acción:

The simplest kind of `if` statement has one test and one action:

```
if conditional_test:  
    do something
```

Puedes poner cualquier prueba condicional en la primera línea y casi cualquier acción en el bloque sangrado que sigue a la prueba. Si el resultado de la prueba condicional es `True`, Python ejecuta el código que sigue a la sentencia `if`. Si el resultado de la prueba es `False`, Python ignora el código que sigue a la sentencia `if`.

You can put any conditional test in the first line and just about any action in the indented block following the test. If the conditional test evaluates to `True`, Python executes the code following the `if` statement. If the test evaluates to `False`, Python ignores the code following the `if` statement.

Supongamos que tenemos una variable que representa la edad de una persona, y queremos saber si esa persona tiene edad suficiente para votar. El siguiente código comprueba si la persona puede votar:

Let's say we have a variable representing a person's age, and we want to know if that person is old enough to vote. The following code tests whether the person can vote:

votar.py

voting.py

```
age = 19  
if age >= 18:  
    print("You are old enough to vote!")
```

Python comprueba si el valor de `age` es mayor o igual que 18 años. Si lo es, Python ejecuta la llamada sangrada a `print()` :

Python checks to see whether the value of `age` is greater than or equal to 18. It is, so Python executes the indented `print()` call:

You are old enough to vote!

La sangría desempeña el mismo papel en las sentencias `if` que en los bucles `for` . Todas las líneas sangradas después de una sentencia `if` se ejecutarán si la prueba se supera, y todo el bloque de líneas sangradas se ignorará si la prueba no se supera.

Indentation plays the same role in `if` statements as it did in `for` loops. All indented lines after an `if` statement will be executed if the test passes, and the entire block of indented lines will be ignored if the test does not pass.

Puedes tener tantas líneas de código como quieras en el bloque que sigue a la sentencia `if` . Añadamos otra línea de salida si la persona tiene edad suficiente para votar, preguntándole si ya se ha registrado para votar:

You can have as many lines of code as you want in the block following the `if` statement. Let's add another line of output if the person is old enough to vote, asking if the individual has registered to vote yet:

```
age = 19
if age >= 18:
    print("You are old enough to vote!")
    print("Have you registered to vote yet?")
```

La prueba condicional pasa, y ambas llamadas a `print()` están sangradas, por lo que se imprimen ambas líneas:

The conditional test passes, and both `print()` calls are indented, so both lines are printed:

You are old enough to vote!
Have you registered to vote yet?

Si el valor de `age` es inferior a 18, este programa no produciría ninguna salida.

If the value of `age` is less than 18, this program would produce no output.

Sentencias if-else (*if-else Statements*)

A menudo, querrás realizar una acción cuando se supere una prueba condicional y una acción diferente en los demás casos. La sintaxis `if - else` de Python lo hace posible. Un bloque `if - else` es similar a una simple sentencia `if`, pero la sentencia `else` te permite definir una acción o conjunto de acciones que se ejecutan cuando falla la prueba condicional.

Often, you'll want to take one action when a conditional test passes and a different action in all other cases. Python's `if - else` syntax makes this possible. An `if - else` block is similar to a simple `if` statement, but the `else` statement allows you to define an action or set of actions that are executed when the conditional test fails.

Mostraremos el mismo mensaje que teníamos anteriormente si la persona tiene edad suficiente para votar, pero esta vez añadiremos un mensaje para quien no tenga edad suficiente para votar:

We'll display the same message we had previously if the person is old enough to vote, but this time we'll add a message for anyone who is not old enough to vote:

```
age = 17
❶ if age >= 18:
    print("You are old enough to vote!")
    print("Have you registered to vote yet?")
❷ else:
    print("Sorry, you are too young to vote.")
    print("Please register to vote as soon as you turn 18!")
```

Si la prueba condicional ❶ pasa, se ejecuta el primer bloque de llamadas sangradas `print()` . Si la prueba evalúa `False` , se ejecuta el bloque `else` ❷ . Como `age` es menor que 18 esta vez, la prueba condicional falla y se ejecuta el código del bloque `else` :

If the conditional test ❶ passes, the first block of indented `print()` calls is executed. If the test evaluates to `False` , the `else` block ❷ is executed. Because `age` is less than 18 this time, the conditional test fails and the code in the `else` block is executed:

```
Sorry, you are too young to vote.  
Please register to vote as soon as you turn 18!
```

Este código funciona porque sólo tiene dos situaciones posibles que evaluar: una persona tiene edad suficiente para votar o no tiene edad suficiente para votar. La estructura `if - else` funciona bien en situaciones en las que quieres que Python ejecute siempre una de las dos acciones posibles. En una cadena `if-else` sencilla como ésta, siempre se ejecutará una de las dos acciones.

This code works because it has only two possible situations to evaluate: a person is either old enough to vote or not old enough to vote. The `if - else` structure works well in situations in which you want Python to always execute one of two possible actions. In a simple `if-else` chain like this, one of the two actions will always be executed.

La cadena `if-elif-else` (*The if-elif-else Chain*)

A menudo, necesitarás probar más de dos situaciones posibles, y para evaluarlas puedes utilizar la sintaxis `if - elif - else` de Python. Python sólo ejecuta un bloque en una cadena `if - elif - else` . Ejecuta cada prueba condicional en orden, hasta que una pasa. Cuando una prueba pasa, se ejecuta el código que sigue a esa prueba y Python se salta el resto de las pruebas.

Often, you'll need to test more than two possible situations, and to evaluate these you can use Python's `if - elif - else` syntax. Python executes only one block in an `if - elif - else` chain. It runs each conditional test in order, until one passes. When a test passes, the code following that test is executed and Python skips the rest of the tests.

Muchas situaciones del mundo real implican más de dos condiciones posibles. Por ejemplo, piensa en un parque de atracciones que cobra tarifas diferentes según la edad:

Many real-world situations involve more than two possible conditions. For example, consider an amusement park that charges different rates for different age groups:

Listado en español:

- Los menores de 4 años entran gratis.
- La entrada para las personas de entre 4 y 18 años cuesta 25 \$.
- La entrada para mayores de 18 años cuesta 40 \$.

Listing in English:

- Admission for anyone under age 4 is free.
- Admission for anyone between the ages of 4 and 18 is \$25.
- Admission for anyone age 18 or older is \$40.

¿Cómo podemos utilizar una sentencia `if` para determinar la tarifa de admisión de una persona? El siguiente código comprueba el grupo de edad de una persona y luego imprime un mensaje con el precio de admisión:

How can we use an `if` statement to determine a person's admission rate? The following code tests for the age group of a person and then prints an admission price message:

`parque_de_atracciones.py`

`amusement_park.py`

```
age = 12
❶ if age < 4:
    print("Your admission cost is $0.")
❷ elif age < 18:
    print("Your admission cost is $25.")
❸ else:
    print("Your admission cost is $40.")
```

La prueba `if` ❶ comprueba si una persona tiene menos de 4 años. Cuando la prueba se supera, se imprime un mensaje apropiado y Python se salta el resto de las pruebas. La línea `elif` ❷ es en realidad otra prueba `if` , que se ejecuta sólo si la prueba anterior falló. En este punto de la cadena, sabemos que la persona tiene al menos 4 años porque la primera prueba falló. Si la persona es menor de 18 años, se imprime un mensaje apropiado y Python se salta el bloque `else` . Si fallan las pruebas `if` y `elif` , Python ejecuta el código del bloque `else` ❸ .

The `if` test ❶ checks whether a person is under 4 years old. When the test passes, an appropriate message is printed and Python skips the rest of the tests. The `elif` line ❷ is really another `if` test, which runs only if the previous test failed. At this point in the chain, we know the person is at least 4 years old because the first test failed. If the person is under 18, an appropriate message is printed and Python skips the `else` block. If both the `if` and `elif` tests fail, Python runs the code in the `else` block ❸ .

En este ejemplo, la prueba `if` ❶ evalúa a `False` , por lo que su bloque de código no se ejecuta. Sin embargo, la prueba `elif` evalúa a `True` (12 es menor que 18), por lo que su código se ejecuta. La salida es una frase, que informa al usuario del coste de la entrada:

In this example the `if` test ❶ evaluates to `False` , so its code block is not executed. However, the `elif` test evaluates to `True` (12 is less than 18) so its code is executed. The output is one sentence, informing the user of the admission cost:

Your admission cost is \$25.

Cualquier edad superior a 17 años provocaría el fallo de las dos primeras pruebas. En estas situaciones, se ejecutaría el bloque `else` y el precio de la entrada sería de 40\$.

Any age greater than 17 would cause the first two tests to fail. In these situations, the `else` block would be executed and the admission price would be \$40.

En lugar de imprimir el precio de la entrada dentro del bloque `if - elif - else` , sería más conciso establecer sólo el precio dentro de la cadena `if - elif - else` y luego tener una única llamada a `print()` que se ejecute después de que se haya evaluado la cadena:

Rather than printing the admission price within the `if - elif - else` block, it would be more concise to set just the price inside the `if - elif - else` chain and then have a single `print()` call that runs after the chain has been evaluated:

```
age = 12

if age < 4:
    price = 0
elif age < 18:
    price = 25
else:
    price = 40

print(f"Your admission cost is ${price}.")
```

Las líneas sangradas fijan el valor de `price` en función de la edad de la persona, como en el ejemplo anterior. Después de que la cadena `if - elif - else` establezca el precio, una llamada independiente sin sangría a `print()` utiliza este valor para mostrar un mensaje informando del precio de admisión de la persona.

The indented lines set the value of `price` according to the person's age, as in the previous example. After the price is set by the `if - elif - else` chain, a separate unindented `print()` call uses this value to display a message reporting the person's admission price.

Este código produce la misma salida que el ejemplo anterior, pero el propósito de la cadena `if - elif - else` es más limitado. En lugar de determinar un precio y mostrar un mensaje, simplemente determina el precio de la entrada. Además de ser más eficaz, este código revisado es más fácil de modificar que el planteamiento original. Para cambiar el texto del mensaje de salida, sólo tendrías que cambiar una llamada a `print()` en lugar de tres llamadas separadas a `print()`.

This code produces the same output as the previous example, but the purpose of the `if - elif - else` chain is narrower. Instead of determining a price and displaying a message, it simply determines the admission price. In addition to being more efficient, this revised code is easier to modify than the original approach. To change the text of the output message, you would need to change only one `print()` call rather than three separate `print()` calls.

Uso de varios bloques elif (Using Multiple elif Blocks)

Puedes utilizar tantos bloques `elif` en tu código como quieras. Por ejemplo, si el parque de atracciones fuera a implementar un descuento para mayores, podrías añadir una prueba condicional más al código para determinar si alguien tiene derecho al descuento para mayores. Digamos que cualquier persona de 65 años o más paga la mitad de la entrada normal, es decir, 20 \$:

You can use as many `elif` blocks in your code as you like. For example, if the amusement park were to implement a discount for seniors, you could add one more conditional test to the code to determine whether someone qualifies for the senior discount. Let's say that anyone 65 or older pays half the regular admission, or \$20:

```
age = 12

if age < 4:
    price = 0
elif age < 18:
    price = 25
elif age < 65:
    price = 40
else:
    price = 20

print(f"Your admission cost is ${price}.")
```

La mayor parte de este código no se modifica. El segundo bloque `elif` comprueba ahora si una persona tiene menos de 65 años antes de asignarle la tarifa de admisión completa de 40 \$. Observa que el valor asignado en el bloque `else` debe cambiarse a 20 \$, porque las únicas edades que llegan a este bloque son las de personas de 65 años o más.

Most of this code is unchanged. The second `elif` block now checks to make sure a person is less than age 65 before assigning them the full admission rate of \$40. Notice that the value assigned in the `else` block needs to be changed to \$20, because the only ages that make it to this block are for people 65 or older.

Omitir el bloque else (*Omitting the else Block*)

Python no requiere un bloque `else` al final de una cadena `if - elif`. A veces, un bloque `else` es útil. Otras veces, es más claro utilizar una sentencia `elif` adicional que capture la condición específica de interés:

Python does not require an `else` block at the end of an `if - elif` chain. Sometimes, an `else` block is useful. Other times, it's clearer to use an additional `elif` statement that catches the specific condition of interest:

```
age = 12

if age < 4:
    price = 0
elif age < 18:
    price = 25
elif age < 65:
    price = 40
elif age >= 65:
    price = 20

print(f"Your admission cost is ${price}.")
```

El bloque final `elif` asigna un precio de 20 \$ cuando la persona tiene 65 años o más, lo que es un poco más claro que el bloque general `else`. Con este cambio, cada bloque de código debe superar una prueba específica para poder ejecutarse.

The final `elif` block assigns a price of \$20 when the person is 65 or older, which is a little clearer than the general `else` block. With this change, every block of code must pass a specific test in order to be executed.

El bloque `else` es una sentencia catchall. Iguala cualquier condición que no haya sido igualada por una prueba específica de `if` o `elif`, y eso a veces puede incluir datos no válidos o incluso maliciosos. Si tienes una condición final específica que estás comprobando, considera la posibilidad de utilizar un bloque `elif` final y omitir el bloque `else`. Como resultado, estarás más seguro de que tu código se ejecutará sólo en las condiciones correctas.

The `else` block is a catchall statement. It matches any condition that wasn't matched by a specific `if` or `elif` test, and that can sometimes include invalid or even malicious data. If you have a specific final condition you're testing for, consider using a final `elif` block and omit the `else` block. As a result, you'll be more confident that your code will run only under the correct conditions.

Probar múltiples condiciones (*Testing Multiple Conditions*)

La cadena `if - elif - else` es potente, pero sólo es apropiado utilizarla cuando sólo necesitas que pase una prueba. En cuanto Python encuentra una prueba que pasa, se salta el resto de las pruebas. Este comportamiento es beneficioso, porque es eficiente y te permite comprobar una condición concreta.

The `if - elif - else` chain is powerful, but it's only appropriate to use when you just need one test to pass. As soon as Python finds one test that passes, it skips the rest of the tests. This behavior is beneficial, because it's efficient and allows you to test for one specific condition.

Sin embargo, a veces es importante comprobar todas las condiciones de interés. En este caso, debes utilizar una serie de sentencias `if` sencillas, sin `elif` ni bloques `else`. Esta técnica tiene sentido cuando más de una condición podría ser `True`, y quieres actuar sobre cada condición que sea `True`.

However, sometimes it's important to check all conditions of interest. In this case, you should use a series of simple `if` statements with no `elif` or `else` blocks. This technique makes sense when more than one condition could be `True`, and you want to act on every condition that is `True`.

Reconsideremos el ejemplo de la pizzería. Si alguien pide una pizza con dos ingredientes, tendrás que asegurarte de incluir ambos ingredientes en su pizza:

Let's reconsider the pizzeria example. If someone requests a two-topping pizza, you'll need to be sure to include both toppings on their pizza:

toppings.py

toppings.py

```
requested_toppings = ['mushrooms', 'extra cheese']
```

```
if 'mushrooms' in requested_toppings:  
    print("Adding mushrooms.")  
❶ if 'pepperoni' in requested_toppings:  
    print("Adding pepperoni.")  
if 'extra cheese' in requested_toppings:  
    print("Adding extra cheese.")
```

```
print("\nFinished making your pizza!")
```

Empezamos con una lista que contiene los ingredientes solicitados. La primera sentencia `if` comprueba si la persona ha pedido champiñones en su pizza. Si es así, se imprime un mensaje confirmando ese ingrediente. La prueba para el pepperoni ❶ es otra sentencia simple `if`, no una sentencia `elif` o `else`, por lo que esta prueba se ejecuta independientemente de si la prueba anterior se ha superado o no. La última sentencia `if` comprueba si se ha pedido queso extra, independientemente de los resultados de las dos primeras pruebas. Estas tres pruebas independientes se ejecutan cada vez que se ejecuta este programa.

We start with a list containing the requested toppings. The first `if` statement checks to see whether the person requested mushrooms on their pizza. If so, a message is printed confirming that topping. The test for pepperoni ❶ is another simple `if` statement, not an `elif` or `else` statement, so this test is run regardless of whether the previous test passed or not. The last `if` statement checks whether extra cheese was requested, regardless of the results from the first two tests. These three independent tests are executed every time this program is run.

Como se evalúan todas las condiciones de este ejemplo, tanto los champiñones como el queso extra se añaden a la pizza:

Because every condition in this example is evaluated, both mushrooms and extra cheese are added to the pizza:

Adding mushrooms.
Adding extra cheese.

Finished making your pizza!

Este código no funcionaría correctamente si utilizáramos un bloque `if - elif - else` , porque el código dejaría de ejecutarse después de pasar una sola prueba. Esto es lo que ocurriría:

This code would not work properly if we used an `if - elif - else` block, because the code would stop running after only one test passes. Here's what that would look like:

```
requested_toppings = ['mushrooms', 'extra cheese']
```

```
if 'mushrooms' in requested_toppings:  
    print("Adding mushrooms.")  
elif 'pepperoni' in requested_toppings:  
    print("Adding pepperoni.")  
elif 'extra cheese' in requested_toppings:  
    print("Adding extra cheese.")
```

```
print("\nFinished making your pizza!")
```

La prueba para `'mushrooms'` es la primera que se supera, por lo que se añaden champiñones a la pizza. Sin embargo, los valores `'extra cheese'` y `'pepperoni'` nunca se comprueban, porque Python no ejecuta ninguna prueba más allá de la primera prueba que pasa en una cadena `if-elif-else` . Se añadirá el primer ingrediente del cliente, pero se omitirán todos los demás:

The test for `'mushrooms'` is the first test to pass, so mushrooms are added to the pizza. However, the values `'extra cheese'` and `'pepperoni'` are never checked, because Python doesn't run any tests beyond the first test that passes in an `if-elif-else` chain. The customer's first topping will be added, but all of their other toppings will be missed:

Adding mushrooms.

Finished making your pizza!

En resumen, si sólo quieres que se ejecute un bloque de código, utiliza una cadena `if - elif - else` . Si necesitas que se ejecute más de un bloque de código, utiliza una serie de sentencias `if` independientes.

In summary, if you want only one block of code to run, use an `if - elif - else` chain. If more than one block of code needs to run, use a series of independent `if` statements.

Pruébalo tú mismo (*Try It Yourself*)

5-3. Colores alienígenas nº 1: Imagina que acaban de derribar a un alienígena en un juego. Crea una variable llamada `alien_color` y asígnale un valor de `'green'` , `'yellow'` , o `'red'` .

5-3. Alien Colors #1: Imagine an alien was just shot down in a game. Create a variable called `alien_color` and assign it a value of `'green'` , `'yellow'` , or `'red'` .

Listado en español:

- Escribe una sentencia `if` para comprobar si el color del alienígena es verde. Si lo es, imprime un mensaje indicando que el jugador acaba de ganar 5 puntos.
- Escribe una versión de este programa que pase la prueba `if` y otra que falle. (La versión que falle no tendrá salida).

Listing in English:

- Write an `if` statement to test whether the alien's color is green. If it is, print a message that the player just earned 5 points.
- Write one version of this program that passes the `if` test and another that fails. (The version that fails will have no output.)

5-4. Colores del alienígena nº 2: Elige un color para un alienígena como hiciste en el Ejercicio 5-3, y escribe una cadena `if - else` .

5-4. Alien Colors #2: Choose a color for an alien as you did in Exercise 5-3, and write an `if - else` chain.

Listado en español:

- Si el color del alienígena es verde, imprime una declaración de que el jugador acaba de ganar 5 puntos por disparar al alienígena.
- Si el color del alienígena no es verde, imprime una declaración que diga que el jugador acaba de ganar 10 puntos.
- Escribe una versión de este programa que ejecute el bloque `if` y otra que ejecute el bloque `else` .

Listing in English:

- If the alien's color is green, print a statement that the player just earned 5 points for shooting the alien.
- If the alien's color isn't green, print a statement that the player just earned 10 points.
- Write one version of this program that runs the `if` block and another that runs the `else` block.

5-5. Colores del Alien nº 3: Convierte tu cadena `if - else` del Ejercicio 5-4 en una cadena `if - elif - else` .

5-5. Alien Colors #3: Turn your `if - else` chain from Exercise 5-4 into an `if - elif - else` chain.

Listado en español:

- Si el alienígena es verde, imprime un mensaje indicando que el jugador ha ganado 5 puntos.
- Si el alienígena es amarillo, imprime un mensaje indicando que el jugador ha ganado 10 puntos.
- Si el alienígena es rojo, imprime un mensaje indicando que el jugador ha ganado 15 puntos.
- Escribe tres versiones de este programa, asegurándote de que cada mensaje se imprime para el alienígena del color apropiado.

Listing in English:

- If the alien is green, print a message that the player earned 5 points.
- If the alien is yellow, print a message that the player earned 10 points.
- If the alien is red, print a message that the player earned 15 points.
- Write three versions of this program, making sure each message is printed for the appropriate color alien.

5-6. Etapas de la vida: Escribe una cadena `if - elif - else` que determine la etapa de la vida de una persona. Establece un valor para la variable `age` , y luego:

5-6. Stages of Life: Write an `if - elif - else` chain that determines a person's stage of life. Set a value for the variable `age` , and then:

Listado en español:

- Si la persona tiene menos de 2 años, imprime un mensaje que indique que la persona es un bebé.
- Si la persona tiene al menos 2 años pero menos de 4, imprime un mensaje indicando que la persona es un niño pequeño.
- Si la persona tiene al menos 4 años pero menos de 13, imprime un mensaje que indique que la persona es un niño.
- Si la persona tiene al menos 13 años pero menos de 20, imprime un mensaje que indique que la persona es un adolescente.
- Si la persona tiene al menos 20 años pero menos de 65, imprime un mensaje que indique que la persona es un adulto.
- Si la persona tiene 65 años o más, imprime un mensaje que indique que la persona es un anciano.

Listing in English:

- If the person is less than 2 years old, print a message that the person is a baby.
- If the person is at least 2 years old but less than 4, print a message that the person is a toddler.
- If the person is at least 4 years old but less than 13, print a message that the person is a kid.
- If the person is at least 13 years old but less than 20, print a message that the person is a teenager.
- If the person is at least 20 years old but less than 65, print a message that the person is an adult.
- If the person is age 65 or older, print a message that the person is an elder.

5-7. Fruta favorita: Haz una lista de tus frutas favoritas y, a continuación, escribe una serie de afirmaciones independientes `if` que comprueben la presencia de determinadas frutas en tu lista.

5-7. Favorite Fruit: Make a list of your favorite fruits, and then write a series of independent `if` statements that check for certain fruits in your list.

Listado en español:

- Haz una lista de tus tres frutas favoritas y llámala `favorite_fruits` .
- Escribe cinco afirmaciones `if` . Cada una de ellas debe comprobar si un determinado tipo de fruta está en tu lista. Si la fruta está en tu lista, el bloque `if` debe imprimir una sentencia como la siguiente *You really like bananas!*

Listing in English:

- Make a list of your three favorite fruits and call it `favorite_fruits` .
 - Write five `if` statements. Each should check whether a certain kind of fruit is in your list. If the fruit is in your list, the `if` block should print a statement, such as *You really like bananas!*
-

Uso de sentencias `if` con listas (*Using if Statements with Lists*)

Puedes hacer cosas interesantes combinando listas y sentencias `if` . Puedes vigilar valores especiales que deban tratarse de forma diferente a otros valores de la lista. Puedes gestionar eficazmente condiciones cambiantes, como la disponibilidad de ciertos artículos en un restaurante a lo largo de un turno. También puedes empezar a probar que tu código funciona como esperas en todas las situaciones posibles.

You can do some interesting work when you combine lists and `if` statements. You can watch for special values that need to be treated differently than other values in the list. You can efficiently manage changing conditions, such as the availability of certain items in a restaurant throughout a shift. You can also begin to prove that your code works as you expect it to in all possible situations.

Comprobación de elementos especiales (*Checking for Special Items*)

Este capítulo comenzó con un ejemplo sencillo que mostraba cómo tratar un valor especial como `'bmw'` , que debía imprimirse en un formato diferente al de otros valores de la lista. Ahora que tienes unos conocimientos básicos sobre las pruebas condicionales y las sentencias `if` , vamos a ver más detenidamente cómo puedes comprobar si hay valores especiales en una lista y manejarlos adecuadamente.

This chapter began with a simple example that showed how to handle a special value like `'bmw'` , which needed to be printed in a different format than other values in the list. Now that you have a basic understanding of conditional tests and `if` statements, let's take a

closer look at how you can watch for special values in a list and handle those values appropriately.

Sigamos con el ejemplo de la pizzería. La pizzería muestra un mensaje cada vez que se añade un ingrediente a la pizza, mientras se está haciendo. El código para esta acción puede escribirse de forma muy eficiente haciendo una lista de los ingredientes que el cliente ha solicitado y utilizando un bucle para anunciar cada ingrediente a medida que se añade a la pizza:

Let's continue with the pizzeria example. The pizzeria displays a message whenever a topping is added to your pizza, as it's being made. The code for this action can be written very efficiently by making a list of toppings the customer has requested and using a loop to announce each topping as it's added to the pizza:

ingredientes.py

toppings.py

```
requested_toppings = ['mushrooms', 'green peppers', 'extra cheese']
```

```
for requested_topping in requested_toppings:  
    print(f'Adding {requested_topping}.')
```

```
print("\nFinished making your pizza!")
```

La salida es sencilla porque este código es un simple bucle for :

The output is straightforward because this code is just a simple for loop:

```
Adding mushrooms.  
Adding green peppers.  
Adding extra cheese.
```

```
Finished making your pizza!
```

¿Pero qué pasa si la pizzería se queda sin pimientos verdes? Una sentencia `if` dentro del bucle `for` puede manejar esta situación adecuadamente:

But what if the pizzeria runs out of green peppers? An `if` statement inside the `for` loop can handle this situation appropriately:

```
requested_toppings = ['mushrooms', 'green peppers', 'extra cheese']

for requested_topping in requested_toppings:
    if requested_topping == 'green peppers':
        print("Sorry, we are out of green peppers right now.")
    else:
        print(f"Adding {requested_topping}.")

print("\nFinished making your pizza!")
```

Esta vez, comprobamos cada elemento solicitado antes de añadirlo a la pizza. La sentencia `if` comprueba si la persona ha solicitado pimientos verdes. Si es así, mostramos un mensaje informándole de por qué no puede tener pimientos verdes. El bloque `else` garantiza que todos los demás ingredientes se añadan a la pizza.

This time, we check each requested item before adding it to the pizza. The `if` statement checks to see if the person requested green peppers. If so, we display a message informing them why they can't have green peppers. The `else` block ensures that all other toppings will be added to the pizza.

La salida muestra que cada aderezo solicitado se gestiona adecuadamente.

The output shows that each requested topping is handled appropriately.

```
Adding mushrooms.
Sorry, we are out of green peppers right now.
Adding extra cheese.
```

```
Finished making your pizza!
```

Comprobar que una lista no está vacía (*Checking That a List Is Not Empty*)

Hemos hecho una suposición sencilla sobre todas las listas con las que hemos trabajado hasta ahora: hemos supuesto que cada lista tiene al menos un elemento. Pronto dejaremos que los usuarios proporcionen la información que se almacena en una lista, por lo que no podremos suponer que una lista tiene algún elemento cada vez que se ejecute un bucle. En esta situación, es útil comprobar si una lista está vacía antes de ejecutar un bucle `for`.

We've made a simple assumption about every list we've worked with so far: we've assumed that each list has at least one item in it. Soon we'll let users provide the information that's stored in a list, so we won't be able to assume that a list has any items in it each time a loop is run. In this situation, it's useful to check whether a list is empty before running a `for` loop.

Como ejemplo, comprobemos si la lista de ingredientes solicitados está vacía antes de construir la pizza. Si la lista está vacía, preguntaremos al usuario y nos aseguraremos de que quiere una pizza normal. Si la lista no está vacía, construiremos la pizza igual que en los ejemplos anteriores:

As an example, let's check whether the list of requested toppings is empty before building the pizza. If the list is empty, we'll prompt the user and make sure they want a plain pizza. If the list is not empty, we'll build the pizza just as we did in the previous examples:

```
requested_toppings = []

if requested_toppings:
    for requested_topping in requested_toppings:
        print(f"Adding {requested_topping}.")
    print("\nFinished making your pizza!")
else:
    print("Are you sure you want a plain pizza?")
```

Esta vez empezamos con una lista vacía de ingredientes solicitados. En lugar de saltar directamente a un bucle `for`, primero hacemos una comprobación rápida. Cuando se utiliza el nombre de una lista en una sentencia `if`, Python devuelve `True` si la lista contiene al menos un elemento; una lista vacía se evalúa como `False`. Si `requested_toppings` supera la prueba condicional, ejecutamos el mismo bucle `for` que utilizamos en el ejemplo anterior. Si la prueba condicional falla, imprimimos un mensaje preguntando al cliente si realmente quiere una pizza normal sin ingredientes.

This time we start out with an empty list of requested toppings. Instead of jumping right into a `for` loop, we do a quick check first. When the name of a list is used in an `if` statement, Python returns `True` if the list contains at least one item; an empty list evaluates to `False`. If `requested_toppings` passes the conditional test, we run the same `for` loop we used in the previous example. If the conditional test fails, we print a message asking the customer if they really want a plain pizza with no toppings.

La lista está vacía en este caso, por lo que la salida pregunta si el usuario realmente quiere una pizza normal:

The list is empty in this case, so the output asks if the user really wants a plain pizza:

Are you sure you want a plain pizza?

Si la lista no está vacía, la salida mostrará cada uno de los ingredientes solicitados añadidos a la pizza.

If the list is not empty, the output will show each requested topping being added to the pizza.

Utilizar varias listas (Using Multiple Lists)

La gente pide casi cualquier cosa, sobre todo cuando se trata de ingredientes de pizza. ¿Y si un cliente quiere patatas fritas en su pizza? Puedes utilizar listas y afirmaciones de `if` para asegurarte de que la información tiene sentido antes de actuar en consecuencia.

People will ask for just about anything, especially when it comes to pizza toppings. What if a customer actually wants french fries on their pizza? You can use lists and `if` statements to make sure your input makes sense before you act on it.

Tengamos cuidado con las peticiones inusuales de ingredientes antes de construir una pizza. El siguiente ejemplo define dos listas. La primera es una lista de ingredientes disponibles en la pizzería, y la segunda es la lista de ingredientes que ha solicitado el

usuario. Esta vez, cada elemento de `requested_toppings` se comprueba con la lista de ingredientes disponibles antes de añadirlo a la pizza:

Let's watch out for unusual topping requests before we build a pizza. The following example defines two lists. The first is a list of available toppings at the pizzeria, and the second is the list of toppings that the user has requested. This time, each item in `requested_toppings` is checked against the list of available toppings before it's added to the pizza:

```
available_toppings = ['mushrooms', 'olives', 'green peppers',  
                     'pepperoni', 'pineapple', 'extra cheese']
```

```
❶ requested_toppings = ['mushrooms', 'french fries', 'extra cheese']
```

```
for requested_topping in requested_toppings:
```

```
❷   if requested_topping in available_toppings:  
       print(f'Adding {requested_topping}.')
```

```
❸   else:  
       print(f'Sorry, we don't have {requested_topping}.')
```

```
print("\nFinished making your pizza!")
```

En primer lugar, definimos una lista de ingredientes disponibles en esta pizzería. Ten en cuenta que puede ser una tupla si la pizzería tiene una selección estable de ingredientes. Después, hacemos una lista de los ingredientes que ha solicitado un cliente. En este ejemplo hay una petición inusual de un ingrediente: 'french fries' ❶. A continuación, recorreremos la lista de ingredientes solicitados. Dentro del bucle, comprobamos si cada cobertura solicitada está en la lista de coberturas disponibles ❷. Si lo está, añadimos ese ingrediente a la pizza. Si el ingrediente solicitado no está en la lista de ingredientes disponibles, se ejecuta el bloque `else` ❸. El bloque `else` imprime un mensaje indicando al usuario qué ingredientes no están disponibles.

First, we define a list of available toppings at this pizzeria. Note that this could be a tuple if the pizzeria has a stable selection of toppings. Then, we make a list of toppings that a customer has requested. There's an unusual request for a topping in this example: 'french fries' ❶. Next we loop through the list of requested toppings. Inside the loop, we check to see if each requested topping is actually in the list of available toppings ❷. If it is, we add that topping to the pizza. If the requested topping is not in the list of available toppings, the `else` block will run ❸. The `else` block prints a message telling the user which toppings are unavailable.

Esta sintaxis de código produce una salida limpia e informativa:

This code syntax produces clean, informative output:

Adding mushrooms.
Sorry, we don't have french fries.
Adding extra cheese.

Finished making your pizza!

En sólo unas pocas líneas de código, ¡hemos gestionado una situación del mundo real con bastante eficacia!

In just a few lines of code, we've managed a real-world situation pretty effectively!

Pruébalo tú mismo (*Try It Yourself*)

5-8. Hola Admin: Haz una lista de cinco o más nombres de usuario, incluyendo el nombre 'admin' . Imagina que estás escribiendo un código que imprimirá un saludo a cada usuario después de que se conecte a un sitio web. Recorre la lista e imprime un saludo a cada usuario.

5-8. Hello Admin: Make a list of five or more usernames, including the name 'admin' . Imagine you are writing code that will print a greeting to each user after they log in to a website. Loop through the list, and print a greeting to each user.

Listado en español:

- Si el nombre de usuario es 'admin' , imprime un saludo especial, como por ejemplo *Hello admin, would you like to see a status report?*
- Si no, imprime un saludo genérico, como *Hello Jaden, thank you for logging in again.*

Listing in English:

- If the username is 'admin' , print a special greeting, such as *Hello admin, would you like to see a status report?*
- Otherwise, print a generic greeting, such as *Hello Jaden, thank you for logging in again.*

5-9. No hay usuarios: Añade una prueba `if` a `hello_admin.py` para asegurarte de que la lista de usuarios no está vacía.

5-9. No Users: Add an `if` test to `hello_admin.py` to make sure the list of users is not empty.

Listado en español:

- Si la lista está vacía, imprime el mensaje *We need to find some users!*
- Elimina todos los nombres de usuario de la lista y asegúrate de que se imprime el mensaje correcto.

Listing in English:

- If the list is empty, print the message *We need to find some users!*
- Remove all of the usernames from your list, and make sure the correct message is printed.

5-10. Comprobación de los nombres de usuario: Haz lo siguiente para crear un programa que simule cómo los sitios web se aseguran de que todo el mundo tiene un nombre de usuario único.

5-10. Checking Usernames: Do the following to create a program that simulates how websites ensure that everyone has a unique username.

Listado en español:

- Haz una lista de cinco o más nombres de usuario llamada `current_users` .
- Haz otra lista de cinco nombres de usuario llamada `new_users` . Asegúrate de que uno o dos de los nuevos nombres de usuario están también en la lista `current_users` .
- Recorre la lista `new_users` para ver si ya se ha utilizado cada nombre de usuario nuevo. Si es así, imprime un mensaje indicando que la persona deberá introducir un nuevo nombre de usuario. Si un nombre de usuario no se ha utilizado, imprime un mensaje diciendo que el nombre de usuario está disponible.

- Asegúrate de que la comparación no distingue entre mayúsculas y minúsculas. Si se ha utilizado 'John' , no debe aceptarse 'JOHN' . (Para ello, tendrás que hacer una copia de `current_users` que contenga las versiones en minúsculas de todos los usuarios existentes).

Listing in English:

- Make a list of five or more usernames called `current_users` .
- Make another list of five usernames called `new_users` . Make sure one or two of the new usernames are also in the `current_users` list.
- Loop through the `new_users` list to see if each new username has already been used. If it has, print a message that the person will need to enter a new username. If a username has not been used, print a message saying that the username is available.
- Make sure your comparison is case insensitive. If 'John' has been used, 'JOHN' should not be accepted. (To do this, you'll need to make a copy of `current_users` containing the lowercase versions of all existing users.)

5-11. Números ordinales: Los números ordinales indican su posición en una lista, como *1st* o *2nd* . La mayoría de los números ordinales terminan en *th* , excepto 1, 2 y 3.

5-11. Ordinal Numbers: Ordinal numbers indicate their position in a list, such as *1st* or *2nd* . Most ordinal numbers end in *th* , except 1, 2, and 3.

Listado en español:

- Almacena los números del 1 al 9 en una lista.
- Recorre la lista en bucle.
- Utiliza una cadena `if - elif - else` dentro del bucle para imprimir el final ordinal adecuado para cada número. Tu salida debe ser "1st 2nd 3rd 4th 5th 6th 7th 8th 9th" , y cada resultado debe estar en una línea separada.

Listing in English:

- Store the numbers 1 through 9 in a list.
 - Loop through the list.
 - Use an `if - elif - else` chain inside the loop to print the proper ordinal ending for each number. Your output should read "1st 2nd 3rd 4th 5th 6th 7th 8th 9th" , and each result should be on a separate line.
-

Cómo dar estilo a tus sentencias if (*Styling Your if Statements*)

En todos los ejemplos de este capítulo has visto buenos hábitos de estilo. La única recomendación que ofrece PEP 8 para dar estilo a las pruebas condicionales es utilizar un solo espacio alrededor de los operadores de comparación, como `==` , `>=` y `<=` . Por ejemplo

In every example in this chapter, you've seen good styling habits. The only recommendation PEP 8 provides for styling conditional tests is to use a single space around comparison operators, such as `==` , `>=` , and `<=` . For example:

`if age < 4:`

es mejor que:

is better than:

`if age<4:`

Este espaciado no afecta a la forma en que Python interpreta tu código; sólo hace que tu código sea más fácil de leer para ti y para los demás.

Such spacing does not affect the way Python interprets your code; it just makes your code easier for you and others to read.

Pruébalo tú mismo (*Try It Yourself*)

5-12. Estilizar **las sentencias if** : Revisa los programas que has escrito en este capítulo y asegúrate de que has dado el estilo adecuado a tus pruebas condicionales.

5-12. Styling **if Statements**: Review the programs you wrote in this chapter, and make sure you styled your conditional tests appropriately.

5-13. Tus ideas: Llegados a este punto, eres un programador más capaz de lo que eras cuando empezaste este libro. Ahora que tienes una mejor idea de cómo se modelan las situaciones del mundo real en los programas, podrías estar pensando en algunos problemas que podrías resolver con tus propios programas. Anota cualquier idea nueva que tengas sobre problemas que podrías querer resolver a medida que tus habilidades de programación sigan mejorando. Piensa en juegos que te gustaría escribir, conjuntos de datos que te gustaría explorar y aplicaciones web que te gustaría crear.

5-13. Your Ideas: At this point, you're a more capable programmer than you were when you started this book. Now that you have a better sense of how real-world situations are modeled in programs, you might be thinking of some problems you could solve with your own programs. Record any new ideas you have about problems you might want to solve as your programming skills continue to improve. Consider games you might want to write, datasets you might want to explore, and web applications you'd like to create.

Resumen (Summary)

En este capítulo aprendiste a escribir pruebas condicionales, que siempre se evalúan como `True` o `False`. Aprendiste a escribir sentencias simples `if`, cadenas `if - else` y cadenas `if - elif - else`. Empezaste a utilizar estas estructuras para identificar determinadas condiciones que necesitas probar y para saber cuándo se han cumplido esas condiciones en tus programas. Aprendiste a tratar determinados elementos de una lista de forma diferente al resto de elementos, sin dejar de utilizar la eficacia de un bucle `for`. También revisaste las recomendaciones de estilo de Python para asegurarte de que tus programas, cada vez más complejos, siguen siendo relativamente fáciles de leer y comprender.

In this chapter you learned how to write conditional tests, which always evaluate to `True` or `False`. You learned to write simple `if` statements, `if - else` chains, and `if - elif - else` chains. You began using these structures to identify particular conditions you need to test and to know when those conditions have been met in your programs. You learned to handle certain items in a list differently than all other items while continuing to utilize the efficiency of a `for` loop. You also revisited Python's style recommendations to ensure that your increasingly complex programs are still relatively easy to read and understand.

En el Capítulo 6 aprenderás sobre los diccionarios de Python. Un diccionario es similar a una lista, pero te permite conectar piezas de información. Aprenderás a construir diccionarios, a recorrerlos en bucle y a utilizarlos en combinación con listas y sentencias `if`. Aprender sobre diccionarios te permitirá modelizar una variedad aún mayor de

situaciones del mundo real.

In Chapter 6 you'll learn about Python's dictionaries. A dictionary is similar to a list, but it allows you to connect pieces of information. You'll learn how to build dictionaries, loop through them, and use them in combination with lists and `if` statements. Learning about dictionaries will enable you to model an even wider variety of real-world situations.

Capítulo 6: Diccionarios (Chapter 6: Dictionaries)

6

Diccionarios (6 Dictionaries)



DESCRIPCIÓN DE LA
IMAGEN (En español):

La imagen muestra una ilustración en blanco y negro de una serpiente que lleva consigo un motor a reacción. La serpiente tiene una expresión desafiante y está enrollada alrededor del motor. El fondo es de color gris, lo que resalta el diseño de la serpiente y el motor. Es un diseño estilizado que combina elementos de naturaleza y tecnología.

En este capítulo aprenderás a utilizar los diccionarios de Python, que te permiten conectar fragmentos de información relacionada. Aprenderás cómo acceder a la información una vez que está en un diccionario y cómo modificarla. Dado que los diccionarios pueden almacenar una cantidad casi ilimitada de información, te mostraré cómo recorrer en bucle los datos de un diccionario. Además, aprenderás a anidar diccionarios dentro de listas, listas dentro de diccionarios e incluso diccionarios dentro de otros diccionarios.

In this chapter you'll learn how to use Python's dictionaries, which allow you to connect pieces of related information. You'll learn how to access the information once it's in a dictionary and how to modify that information. Because dictionaries can store an almost limitless amount of information, I'll show you how to loop through the data in a dictionary. Additionally, you'll learn to nest dictionaries inside lists, lists inside dictionaries, and even dictionaries inside other dictionaries.

Comprender los diccionarios te permitirá modelar con mayor precisión diversos objetos del mundo real. Podrás crear un diccionario que represente a una persona y almacenar toda la información que quieras sobre ella. Podrás almacenar su nombre, edad, ubicación,

profesión y cualquier otro aspecto de una persona que puedas describir. Podrás almacenar dos tipos cualesquiera de información que puedan emparejarse, como una lista de palabras y sus significados, una lista de nombres de personas y sus números favoritos, una lista de montañas y sus elevaciones, etc.

Understanding dictionaries allows you to model a variety of real-world objects more accurately. You'll be able to create a dictionary representing a person and then store as much information as you want about that person. You can store their name, age, location, profession, and any other aspect of a person you can describe. You'll be able to store any two kinds of information that can be matched up, such as a list of words and their meanings, a list of people's names and their favorite numbers, a list of mountains and their elevations, and so forth.

Un diccionario sencillo (*A Simple Dictionary*)

Considera un juego con extraterrestres que pueden tener diferentes colores y valores de puntos. Este sencillo diccionario almacena información sobre un alienígena concreto:

Consider a game featuring aliens that can have different colors and point values. This simple dictionary stores information about a particular alien:

alien.py

alien.py

```
alien_0 = {'color': 'green', 'points': 5}
```

```
print(alien_0['color'])  
print(alien_0['points'])
```

El diccionario `alien_0` almacena el color y el valor en puntos del alienígena. Las dos últimas líneas acceden a esa información y la muestran, como se muestra aquí:

The dictionary `alien_0` stores the alien's color and point value. The last two lines access and display that information, as shown here:

```
green  
5
```

Como ocurre con la mayoría de los nuevos conceptos de programación, utilizar diccionarios requiere práctica. Una vez que hayas trabajado un poco con los diccionarios, verás la eficacia con la que pueden modelar situaciones del mundo real.

As with most new programming concepts, using dictionaries takes practice. Once you've worked with dictionaries for a bit, you'll see how effectively they can model real-world situations.

Trabajar con diccionarios (*Working with Dictionaries*)

Un *dictionary* en Python es una colección de *key-value pairs*. Cada *key* está conectado a un valor, y puedes utilizar una clave para acceder al valor asociado a esa clave. El valor de una clave puede ser un número, una cadena, una lista o incluso otro diccionario. De hecho, puedes utilizar cualquier objeto que puedas crear en Python como valor en un diccionario.

A *dictionary* in Python is a collection of *key-value pairs*. Each *key* is connected to a value, and you can use a key to access the value associated with that key. A key's value can be a number, a string, a list, or even another dictionary. In fact, you can use any object that you can create in Python as a value in a dictionary.

En Python, un diccionario se encierra entre llaves (`{ }`) con una serie de pares clave-valor dentro de las llaves, como se muestra en el ejemplo anterior:

In Python, a dictionary is wrapped in braces (`{ }`) with a series of key-value pairs inside the braces, as shown in the earlier example:

```
alien_0 = {'color': 'green', 'points': 5}
```

Un *key-value pair* es un conjunto de valores asociados entre sí. Cuando proporcionas una clave, Python devuelve el valor asociado a esa clave. Cada clave está unida a su valor por dos puntos, y los pares clave-valor individuales están separados por comas. Puedes almacenar tantos pares clave-valor como quieras en un diccionario.

A *key-value pair* is a set of values associated with each other. When you provide a key, Python returns the value associated with that key. Every key is connected to its value by a colon, and individual key-value pairs are separated by commas. You can store as many key-value pairs as you want in a dictionary.

El diccionario más sencillo tiene exactamente un par clave-valor, como se muestra en esta versión modificada del diccionario `alien_0` :

The simplest dictionary has exactly one key-value pair, as shown in this modified version of the `alien_0` dictionary:

```
alien_0 = {'color': 'green'}
```

Este diccionario almacena una información sobre `alien_0` : el color del extraterrestre. La cadena `'color'` es una clave de este diccionario, y su valor asociado es `'green'` .

This dictionary stores one piece of information about `alien_0` : the alien's color. The string `'color'` is a key in this dictionary, and its associated value is `'green'` .

Acceder a los valores de un diccionario (*Accessing Values in a Dictionary*)

Para obtener el valor asociado a una clave, indica el nombre del diccionario y, a continuación, coloca la clave dentro de un conjunto de corchetes, como se muestra aquí:

To get the value associated with a key, give the name of the dictionary and then place the key inside a set of square brackets, as shown here:

alien.py

alien.py

```
alien_0 = {'color': 'green'}  
print(alien_0['color'])
```

Esto devuelve el valor asociado a la clave 'color' del diccionario alien_0 :

This returns the value associated with the key 'color' from the dictionary alien_0 :

green

Puedes tener un número ilimitado de pares clave-valor en un diccionario. Por ejemplo, aquí tienes el diccionario original alien_0 con dos pares clave-valor:

You can have an unlimited number of key-value pairs in a dictionary. For example, here's the original alien_0 dictionary with two key-value pairs:

```
alien_0 = {'color': 'green', 'points': 5}
```

Ahora puedes acceder al color o al valor en puntos de alien_0 . Si un jugador derriba a este alienígena, puedes consultar cuántos puntos debe ganar utilizando un código como éste:

Now you can access either the color or the point value of alien_0 . If a player shoots down this alien, you can look up how many points they should earn using code like this:

```
alien_0 = {'color': 'green', 'points': 5}
```

```
new_points = alien_0['points']  
print(f"You just earned {new_points} points!")
```

Una vez definido el diccionario, extraemos del diccionario el valor asociado a la clave 'points' . Este valor se asigna a la variable new_points . La última línea imprime una declaración sobre cuántos puntos acaba de ganar el jugador:

Once the dictionary has been defined, we pull the value associated with the key 'points' from the dictionary. This value is then assigned to the variable new_points . The last line prints a statement about how many points the player just earned:

You just earned 5 points!

Si ejecutas este código cada vez que se derriba a un alienígena, se recuperará el valor en puntos del alienígena.

If you run this code every time an alien is shot down, the alien's point value will be retrieved.

Añadir nuevos pares clave-valor *(Adding New Key-Value Pairs)*

Los diccionarios son estructuras dinámicas, y puedes añadir nuevos pares clave-valor a un diccionario en cualquier momento. Para añadir un nuevo par clave-valor, debes indicar el nombre del diccionario seguido de la nueva clave entre corchetes, junto con el nuevo valor.

Dictionaries are dynamic structures, and you can add new key-value pairs to a dictionary at any time. To add a new key-value pair, you would give the name of the dictionary followed by the new key in square brackets, along with the new value.

Vamos a añadir dos nuevos datos al diccionario `alien_0` : las coordenadas `x` - `y` - del alienígena, que nos ayudarán a mostrar al alienígena en una posición determinada de la pantalla. Vamos a colocar al robot en el borde izquierdo de la pantalla, 25 píxeles por debajo de la parte superior. Como las coordenadas de pantalla suelen empezar en la esquina superior izquierda de la pantalla, colocaremos al alienígena en el borde izquierdo de la pantalla poniendo la coordenada `x` a 0 y a 25 píxeles de la parte superior poniendo su coordenada `y` a 25 positivo, como se muestra aquí:

Let's add two new pieces of information to the `alien_0` dictionary: the alien's `x` - and `y` -coordinates, which will help us display the alien at a particular position on the screen. Let's place the alien on the left edge of the screen, 25 pixels down from the top. Because screen coordinates usually start at the upper-left corner of the screen, we'll place the alien on the left edge of the screen by setting the `x` -coordinate to 0 and 25 pixels from the top by setting its `y` -coordinate to positive 25, as shown here:

alien.py

alien.py

```
alien_0 = {'color': 'green', 'points': 5}  
print(alien_0)
```

```
alien_0['x_position'] = 0  
alien_0['y_position'] = 25  
print(alien_0)
```

Empezamos definiendo el mismo diccionario con el que hemos estado trabajando. A continuación, imprimimos este diccionario, mostrando una instantánea de su información. A continuación, añadimos un nuevo par clave-valor al diccionario: la clave 'x_position' y el valor 0 . Hacemos lo mismo con la clave 'y_position' . Cuando imprimimos el diccionario modificado, vemos los dos pares clave-valor adicionales:

We start by defining the same dictionary that we've been working with. We then print this dictionary, displaying a snapshot of its information. Next, we add a new key-value pair to the dictionary: the key 'x_position' and the value 0 . We do the same for the key 'y_position' . When we print the modified dictionary, we see the two additional key-value pairs:

```
{'color': 'green', 'points': 5}  
{'color': 'green', 'points': 5, 'x_position': 0, 'y_position': 25}
```

La versión final del diccionario contiene cuatro pares clave-valor. Los dos originales especifican el color y el valor del punto, y dos más especifican la posición del alienígena.

The final version of the dictionary contains four key-value pairs. The original two specify color and point value, and two more specify the alien's position.

Los diccionarios conservan el orden en que fueron definidos. Cuando imprimas un diccionario o hagas un bucle a través de sus elementos, verás los elementos en el mismo orden en que se añadieron al diccionario.

Dictionaries retain the order in which they were defined. When you print a dictionary or loop through its elements, you will see the elements in the same order they were added to the dictionary.

Empezar con un diccionario vacío (*Starting with an Empty Dictionary*)

A veces es conveniente, o incluso necesario, empezar con un diccionario vacío y luego añadirle cada nuevo elemento. Para empezar a rellenar un diccionario vacío, define un diccionario con un conjunto de llaves vacío y luego añade cada par clave-valor en su propia línea. Por ejemplo, así se construye el diccionario `alien_0` con este método:

It's sometimes convenient, or even necessary, to start with an empty dictionary and then add each new item to it. To start filling an empty dictionary, define a dictionary with an empty set of braces and then add each key-value pair on its own line. For example, here's how to build the `alien_0` dictionary using this approach:

alien.py

alien.py

```
alien_0 = {}
```

```
alien_0['color'] = 'green'
```

```
alien_0['points'] = 5
```

```
print(alien_0)
```

Primero definimos un diccionario `alien_0` vacío, y luego le añadimos los valores de color y punto. El resultado es el diccionario que hemos utilizado en los ejemplos anteriores:

We first define an empty `alien_0` dictionary, and then add color and point values to it. The result is the dictionary we've been using in previous examples:

```
{'color': 'green', 'points': 5}
```

Normalmente, utilizarás diccionarios vacíos cuando almacenes datos proporcionados por el usuario en un diccionario o cuando escribas código que genere un gran número de pares clave-valor automáticamente.

Typically, you'll use empty dictionaries when storing user-supplied data in a dictionary or when writing code that generates a large number of key-value pairs automatically.

Modificar valores de un diccionario *(Modifying Values in a Dictionary)*

Para modificar un valor de un diccionario, indica el nombre del diccionario con la clave entre corchetes y, a continuación, el nuevo valor que quieres asociar a esa clave. Por ejemplo, considera un alienígena que cambia de verde a amarillo a medida que avanza la partida:

To modify a value in a dictionary, give the name of the dictionary with the key in square brackets and then the new value you want associated with that key. For example, consider an alien that changes from green to yellow as a game progresses:

alien.py

alien.py

```
alien_0 = {'color': 'green'}  
print(f"The alien is {alien_0['color']}".)
```

```
alien_0['color'] = 'yellow'  
print(f"The alien is now {alien_0['color']}".)
```

Primero definimos un diccionario para `alien_0` que sólo contiene el color del alienígena; luego cambiamos el valor asociado a la clave `'color'` por `'yellow'` . La salida muestra que, efectivamente, el alienígena ha cambiado de verde a amarillo:

We first define a dictionary for `alien_0` that contains only the alien's color; then we change the value associated with the key 'color' to 'yellow'. The output shows that the alien has indeed changed from green to yellow:

The alien is green.

The alien is now yellow.

Para un ejemplo más interesante, rastreemos la posición de un alienígena que puede moverse a distintas velocidades. Almacenaremos un valor que represente la velocidad actual del robot y lo utilizaremos para determinar cuánto debe moverse hacia la derecha:

For a more interesting example, let's track the position of an alien that can move at different speeds. We'll store a value representing the alien's current speed and then use it to determine how far to the right the alien should move:

```
alien_0 = {'x_position': 0, 'y_position': 25, 'speed': 'medium'}
print(f"Original position: {alien_0['x_position']}")
```

```
# Move the alien to the right.
```

```
# Determine how far to move the alien based on its current speed.
```

```
❶ if alien_0['speed'] == 'slow':
```

```
    x_increment = 1
```

```
elif alien_0['speed'] == 'medium':
```

```
    x_increment = 2
```

```
else:
```

```
    # This must be a fast alien.
```

```
    x_increment = 3
```

```
# The new position is the old position plus the increment.
```

```
❷ alien_0['x_position'] = alien_0['x_position'] + x_increment
```

```
print(f"New position: {alien_0['x_position']}")
```

Empezamos definiendo un alienígena con una posición inicial de `x` y una posición de `y`, y una velocidad de 'medium'. Hemos omitido los valores de color y punto para simplificar, pero este ejemplo funcionaría igual si incluyeras también esos pares clave-valor. También imprimimos el valor original de `x_position` para ver cuánto se desplaza el extraterrestre hacia la derecha.

We start by defining an alien with an initial `x` position and `y` position, and a speed of 'medium' . We've omitted the color and point values for the sake of simplicity, but this example would work the same way if you included those key-value pairs as well. We also print the original value of `x_position` to see how far the alien moves to the right.

Una cadena `if - elif - else` determina a qué distancia debe moverse el alienígena hacia la derecha, y asigna este valor a la variable `x_increment` ❶ . Si la velocidad del alienígena es 'slow' , se desplaza una unidad a la derecha; si la velocidad es 'medium' , se desplaza dos unidades a la derecha; y si es 'fast' , se desplaza tres unidades a la derecha. Una vez calculado el incremento, se suma al valor de `x_position` ❷ , y el resultado se almacena en el diccionario `x_position` .

An `if - elif - else` chain determines how far the alien should move to the right, and assigns this value to the variable `x_increment` ❶ . If the alien's speed is 'slow' , it moves one unit to the right; if the speed is 'medium' , it moves two units to the right; and if it's 'fast' , it moves three units to the right. Once the increment has been calculated, it's added to the value of `x_position` ❷ , and the result is stored in the dictionary's `x_position` .

Como se trata de un alienígena de velocidad media, su posición se desplaza dos unidades a la derecha:

Because this is a medium-speed alien, its position shifts two units to the right:

Original x-position: 0

New x-position: 2

Esta técnica es bastante chula: cambiando un valor del diccionario del alienígena, puedes cambiar el comportamiento general del alienígena. Por ejemplo, para convertir a este alienígena de velocidad media en un alienígena rápido, añadirías esta línea:

This technique is pretty cool: by changing one value in the alien's dictionary, you can change the overall behavior of the alien. For example, to turn this medium-speed alien into a fast alien, you would add this line:

```
alien_0['speed'] = 'fast'
```

El bloque `if - elif - else` asignaría entonces un valor mayor a `x_increment` la próxima vez que se ejecute el código.

The `if - elif - else` block would then assign a larger value to `x_increment` the next time the code runs.

Eliminar pares clave-valor (*Removing Key-Value Pairs*)

Cuando ya no necesitas una información almacenada en un diccionario, puedes utilizar la sentencia `del` para eliminar por completo un par clave-valor. Todo lo que necesita `del` es el nombre del diccionario y la clave que quieres eliminar.

When you no longer need a piece of information that's stored in a dictionary, you can use the `del` statement to completely remove a key-value pair. All `del` needs is the name of the dictionary and the key that you want to remove.

Por ejemplo, vamos a eliminar la clave `'points'` del diccionario `alien_0`, junto con su valor:

For example, let's remove the key `'points'` from the `alien_0` dictionary, along with its value:

alien.py

alien.py

```
alien_0 = {'color': 'green', 'points': 5}
print(alien_0)
```

```
❶ del alien_0['points']
print(alien_0)
```

La sentencia del ❶ indica a Python que elimine la clave 'points' del diccionario alien_0 y que elimine también el valor asociado a esa clave. La salida muestra que la clave 'points' y su valor 5 se eliminan del diccionario, pero el resto del diccionario no se ve afectado:

The del statement ❶ tells Python to delete the key 'points' from the dictionary alien_0 and to remove the value associated with that key as well. The output shows that the key 'points' and its value of 5 are deleted from the dictionary, but the rest of the dictionary is unaffected:

```
{'color': 'green', 'points': 5}
{'color': 'green'}
```

Nota (Note)

Ten en cuenta que el par clave-valor borrado se elimina permanentemente.

Be aware that the deleted key-value pair is removed permanently.

Un diccionario de objetos similares (A Dictionary of Similar Objects)

El ejemplo anterior consistía en almacenar distintos tipos de información sobre un objeto, un alienígena en un juego. También puedes utilizar un diccionario para almacenar un tipo de información sobre muchos objetos. Por ejemplo, supongamos que quieres hacer una encuesta a varias personas y preguntarles cuál es su lenguaje de programación favorito. Un diccionario es útil para almacenar los resultados de una encuesta sencilla, como ésta:

The previous example involved storing different kinds of information about one object, an alien in a game. You can also use a dictionary to store one kind of information about many objects. For example, say you want to poll a number of people and ask them what their favorite programming language is. A dictionary is useful for storing the results of a simple poll, like this:

lenguajes_favoritos.py

favorite_languages.py

```
favorite_languages = {  
    'jen': 'python',  
    'sarah': 'c',  
    'edward': 'rust',  
    'phil': 'python',  
}
```

Como puedes ver, hemos dividido un diccionario más grande en varias líneas. Cada clave es el nombre de una persona que ha respondido a la encuesta, y cada valor es su elección de idioma. Cuando sepas que vas a necesitar más de una línea para definir un diccionario, pulsa INTRO después de la llave de apertura. A continuación, aplica una sangría de un nivel (cuatro espacios) a la línea siguiente y escribe el primer par clave-valor, seguido de una coma. A partir de este momento, cuando pulses INTRO, tu editor de texto debería sangrar automáticamente todos los pares clave-valor siguientes para que coincidan con el primer par clave-valor.

As you can see, we've broken a larger dictionary into several lines. Each key is the name of a person who responded to the poll, and each value is their language choice. When you know you'll need more than one line to define a dictionary, press ENTER after the opening brace. Then indent the next line one level (four spaces) and write the first key-value pair, followed by a comma. From this point forward when you press ENTER, your text editor should automatically indent all subsequent key-value pairs to match the first key-value pair.

Cuando hayas terminado de definir el diccionario, añade un corchete de cierre en una nueva línea después del último par clave-valor, y aplica una sangría de un nivel para que se alinee con las claves del diccionario. Es una buena práctica incluir también una coma después del último par clave-valor, para que estés preparado para añadir un nuevo par clave-valor en la línea siguiente.

Once you've finished defining the dictionary, add a closing brace on a new line after the last key-value pair, and indent it one level so it aligns with the keys in the dictionary. It's good practice to include a comma after the last key-value pair as well, so you're ready to add a new key-value pair on the next line.

Nota (Note)

La mayoría de los editores tienen alguna funcionalidad que te ayuda a dar formato a las listas y diccionarios extensos de forma similar a este ejemplo. También existen otras formas aceptables de dar formato a los diccionarios largos, por lo que es posible que veas un formato ligeramente distinto en tu editor o en otras fuentes.

Most editors have some functionality that helps you format extended lists and dictionaries in a similar manner to this example. Other acceptable ways to format long dictionaries are available as well, so you may see slightly different formatting in your editor, or in other sources.

Para utilizar este diccionario, dado el nombre de una persona que haya participado en la encuesta, puedes buscar fácilmente su idioma favorito:

To use this dictionary, given the name of a person who took the poll, you can easily look up their favorite language:

idiomas_favoritos.py

favorite_languages.py

```
favorite_languages = {  
    'jen': 'python',  
    'sarah': 'c',  
    'edward': 'rust',  
    'phil': 'python',  
}
```

```
❶ language = favorite_languages['sarah'].title()  
print(f"Sarah's favorite language is {language}.")
```

Para ver qué idioma eligió Sarah, pedimos el valor en:

To see which language Sarah chose, we ask for the value at:

```
favorite_languages['sarah']
```

Utilizamos esta sintaxis para sacar la lengua favorita de Sarah del diccionario ❶ y asignarla a la variable `language`. Crear una nueva variable aquí hace que la llamada a `print()` sea mucho más limpia. El resultado muestra el idioma favorito de Sara:

We use this syntax to pull Sarah's favorite language from the dictionary ❶ and assign it to the variable `language`. Creating a new variable here makes for a much cleaner `print()` call. The output shows Sarah's favorite language:

Sarah's favorite language is C.

Podrías utilizar esta misma sintaxis con cualquier individuo representado en el diccionario.

You could use this same syntax with any individual represented in the dictionary.

Utilizar `get()` para acceder a los valores *(Using `get()` to Access Values)*

Utilizar claves entre corchetes para recuperar el valor que te interesa de un diccionario puede causar un problema potencial: si la clave que pides no existe, obtendrás un error.

Using keys in square brackets to retrieve the value you're interested in from a dictionary might cause one potential problem: if the key you ask for doesn't exist, you'll get an error.

Veamos qué ocurre cuando pides el valor en puntos de un alienígena que no tiene ningún valor en puntos establecido:

Let's see what happens when you ask for the point value of an alien that doesn't have a point value set:

alien_sin_puntos.py

alien_no_points.py

```
alien_0 = {'color': 'green', 'speed': 'slow'}  
print(alien_0['points'])
```

Esto da como resultado un rastreo, que muestra un `KeyError` :

This results in a traceback, showing a `KeyError` :

```
Traceback (most recent call last):  
  File "alien_no_points.py", line 2, in <module>  
    print(alien_0['points'])  
~~~~~^~~~~~
```

`KeyError: 'points'`

En el Capítulo 10 aprenderás más sobre cómo tratar errores como éste en general. En el caso concreto de los diccionarios, puedes utilizar el método `get()` para establecer un valor por defecto que se devolverá si la clave solicitada no existe.

You'll learn more about how to handle errors like this in general in Chapter 10 . For dictionaries specifically, you can use the `get()` method to set a default value that will be returned if the requested key doesn't exist.

El método `get()` requiere una clave como primer argumento. Como segundo argumento opcional, puedes pasar el valor que se devolverá si la clave no existe:

The `get()` method requires a key as a first argument. As a second optional argument, you can pass the value to be returned if the key doesn't exist:

```
alien_0 = {'color': 'green', 'speed': 'slow'}
```

```
point_value = alien_0.get('points', 'No point value assigned.')  
print(point_value)
```

Si la clave 'points' existe en el diccionario, obtendrás el valor correspondiente. Si no existe, obtendrás el valor por defecto. En este caso, `points` no existe, y obtenemos un mensaje limpio en lugar de un error:

If the key 'points' exists in the dictionary, you'll get the corresponding value. If it doesn't, you get the default value. In this case, `points` doesn't exist, and we get a clean message instead of an error:

No point value assigned.

Si existe la posibilidad de que la clave que pides no exista, considera la posibilidad de utilizar el método `get()` en lugar de la notación de corchetes.

If there's a chance the key you're asking for might not exist, consider using the `get()` method instead of the square bracket notation.

Nota (Note)

Si omites el segundo argumento en la llamada a `get()` y la clave no existe, Python devolverá el valor `None`. El valor especial `None` significa "no existe ningún valor". No es un error: es un valor especial para indicar la ausencia de un valor. Verás más usos de `None` en el Capítulo 8.

If you leave out the second argument in the call to `get()` and the key doesn't exist, Python will return the value `None`. The special value `None` means "no value exists." This is not an error: it's a special value meant to indicate the absence of a value. You'll see more uses for `None` in Chapter 8.

Pruébalo tú mismo (*Try It Yourself*)

6-1. Persona: Utiliza un diccionario para almacenar información sobre una persona que conozcas. Almacena su nombre, apellidos, edad y la ciudad en la que vive. Deberías tener claves como `first_name` , `last_name` , `age` , y `city` . Imprime cada dato almacenado en tu diccionario.

6-1. Person: Use a dictionary to store information about a person you know. Store their first name, last name, age, and the city in which they live. You should have keys such as `first_name` , `last_name` , `age` , and `city` . Print each piece of information stored in your dictionary.

6-2. Números favoritos: Utiliza un diccionario para almacenar los números favoritos de la gente. Piensa en cinco nombres y utilízalos como claves en tu diccionario. Piensa en un número favorito para cada persona, y almacena cada uno como un valor en tu diccionario. Imprime el nombre de cada persona y su número favorito. Para divertirte aún más, sondea a unos cuantos amigos y obtén datos reales para tu programa.

6-2. Favorite Numbers: Use a dictionary to store people's favorite numbers. Think of five names, and use them as keys in your dictionary. Think of a favorite number for each person, and store each as a value in your dictionary. Print each person's name and their favorite number. For even more fun, poll a few friends and get some actual data for your program.

6-3. Glosario: Un diccionario Python puede utilizarse para modelar un diccionario real. Sin embargo, para evitar confusiones, vamos a llamarlo glosario.

6-3. Glossary: A Python dictionary can be used to model an actual dictionary. However, to avoid confusion, let's call it a glossary.

Listado en español:

- Piensa en cinco palabras de programación que hayas aprendido en los capítulos anteriores. Utiliza estas palabras como claves de tu glosario, y almacena sus significados como valores.

- Imprime cada palabra y su significado como salida con un formato ordenado. Puedes imprimir la palabra seguida de dos puntos y luego su significado, o imprimir la palabra en una línea y luego imprimir su significado sangrado en una segunda línea. Utiliza el carácter de nueva línea (`\n`) para insertar una línea en blanco entre cada par palabra-significado.

Listing in English:

- Think of five programming words you've learned about in the previous chapters. Use these words as the keys in your glossary, and store their meanings as values.
 - Print each word and its meaning as neatly formatted output. You might print the word followed by a colon and then its meaning, or print the word on one line and then print its meaning indented on a second line. Use the newline character (`\n`) to insert a blank line between each word-meaning pair in your output.
-

Recorrer un diccionario *(Looping Through a Dictionary)*

Un diccionario Python puede contener unos pocos pares clave-valor o millones de pares. Como un diccionario puede contener grandes cantidades de datos, Python te permite recorrerlo en bucle. Los diccionarios se pueden utilizar para almacenar información de diversas formas, por lo que existen varias maneras de recorrerlos. Puedes recorrer todos los pares clave-valor de un diccionario, sus claves o sus valores.

A single Python dictionary can contain just a few key-value pairs or millions of pairs. Because a dictionary can contain large amounts of data, Python lets you loop through a dictionary. Dictionaries can be used to store information in a variety of ways; therefore, several different ways exist to loop through them. You can loop through all of a dictionary's key-value pairs, through its keys, or through its values.

Recorrer todos los pares clave-valor *(Looping Through All Key-Value Pairs)*

Antes de explorar los distintos enfoques del bucle, consideremos un nuevo diccionario diseñado para almacenar información sobre un usuario en un sitio web. El siguiente diccionario almacenaría el nombre de usuario, el nombre y los apellidos de una persona:

Before we explore the different approaches to looping, let's consider a new dictionary designed to store information about a user on a website. The following dictionary would store one person's username, first name, and last name:

usuario.py

user.py

```
user_0 = {  
    'username': 'efermi',  
    'first': 'enrico',  
    'last': 'fermi',  
}
```

Puedes acceder a cualquier dato de `user_0` basándote en lo que ya has aprendido en este capítulo. Pero, ¿y si quisieras ver todo lo almacenado en el diccionario de este usuario? Para ello, podrías recorrer el diccionario utilizando un bucle `for` :

You can access any single piece of information about `user_0` based on what you've already learned in this chapter. But what if you wanted to see everything stored in this user's dictionary? To do so, you could loop through the dictionary using a `for` loop:

```
user_0 = {  
    'username': 'efermi',  
    'first': 'enrico',  
    'last': 'fermi',  
}
```

```
for key, value in user_0.items():  
    print(f"\nKey: {key}")  
    print(f"Value: {value}")
```

Para escribir un bucle `for` para un diccionario, crea nombres para las dos variables que contendrán la clave y el valor de cada par clave-valor. Puedes elegir los nombres que quieras para estas dos variables. Este código funcionaría igual de bien si hubieras utilizado abreviaturas para los nombres de las variables, como ésta:

To write a `for` loop for a dictionary, you create names for the two variables that will hold the key and value in each key-value pair. You can choose any names you want for these two variables. This code would work just as well if you had used abbreviations for the variable names, like this:

```
for k, v in user_0.items()
```

La segunda mitad de la sentencia `for` incluye el nombre del diccionario seguido del método `items()`, que devuelve una secuencia de pares clave-valor. A continuación, el bucle `for` asigna cada uno de estos pares a las dos variables proporcionadas. En el ejemplo anterior, utilizamos las variables para imprimir cada `key`, seguido del `value` asociado. El `"\n"` de la primera llamada a `print()` garantiza que se inserte una línea en blanco antes de cada par clave-valor en la salida:

The second half of the `for` statement includes the name of the dictionary followed by the method `items()`, which returns a sequence of key-value pairs. The `for` loop then assigns each of these pairs to the two variables provided. In the preceding example, we use the variables to print each `key`, followed by the associated `value`. The `"\n"` in the first `print()` call ensures that a blank line is inserted before each key-value pair in the output:

Key: username

Value: efermi

Key: first

Value: enrico

Key: last

Value: fermi

El bucle a través de todos los pares clave-valor funciona especialmente bien para diccionarios como el ejemplo *favorite_languages.py* de la página 96, que almacena el mismo tipo de información para muchas claves diferentes. Si recorres el diccionario `favorite_languages`, obtendrás el nombre de cada persona del diccionario y su lenguaje de programación favorito. Como las claves siempre se refieren al nombre de una persona y el valor siempre es un lenguaje, utilizaremos las variables `name` y `language` en el bucle en lugar de `key` y `value`. Así será más fácil seguir lo que ocurre dentro del bucle:

Looping through all key-value pairs works particularly well for dictionaries like the *favorite_languages.py* example on page 96, which stores the same kind of information for many different keys. If you loop through the `favorite_languages` dictionary, you get the name of each person in the dictionary and their favorite programming language. Because the keys always refer to a person's name and the value is always a language, we'll use the

variables `name` and `language` in the loop instead of `key` and `value` . This will make it easier to follow what's happening inside the loop:

idiomas_favoritos.py

favorite_languages.py

```
favorite_languages = {  
    'jen': 'python',  
    'sarah': 'c',  
    'edward': 'rust',  
    'phil': 'python',  
}
```

```
for name, language in favorite_languages.items():  
    print(f'{name.title()}'s favorite language is {language.title()}."
```

Este código le dice a Python que recorra cada par clave-valor del diccionario. A medida que recorre cada par, la clave se asigna a la variable `name` , y el valor se asigna a la variable `language` . Estos nombres descriptivos hacen que sea mucho más fácil ver lo que hace la llamada a `print()` .

This code tells Python to loop through each key-value pair in the dictionary. As it works through each pair the key is assigned to the variable `name` , and the value is assigned to the variable `language` . These descriptive names make it much easier to see what the `print()` call is doing.

Ahora, en unas pocas líneas de código, podemos mostrar toda la información del sondeo:

Now, in just a few lines of code, we can display all of the information from the poll:

Jen's favorite language is Python.
Sarah's favorite language is C.

Edward's favorite language is Rust.
Phil's favorite language is Python.

Este tipo de bucle funcionaría igual de bien si nuestro diccionario almacenara los resultados de encuestar a mil o incluso a un millón de personas.

This type of looping would work just as well if our dictionary stored the results from polling a thousand or even a million people.

Recorrer todas las claves de un diccionario *(Looping Through All the Keys in a Dictionary)*

El método `keys()` es útil cuando no necesitas trabajar con todos los valores de un diccionario. Hagamos un bucle en el diccionario `favorite_languages` e imprimamos los nombres de todas las personas que han participado en la encuesta:

The `keys()` method is useful when you don't need to work with all of the values in a dictionary. Let's loop through the `favorite_languages` dictionary and print the names of everyone who took the poll:

```
favorite_languages = {  
    'jen': 'python',  
    'sarah': 'c',  
    'edward': 'rust',  
    'phil': 'python',  
}
```

```
for name in favorite_languages.keys():  
    print(name.title())
```

Este bucle `for` le dice a Python que extraiga todas las claves del diccionario `favorite_languages` y las asigne de una en una a la variable `name`. La salida muestra los nombres de todos los participantes en la encuesta:

This `for` loop tells Python to pull all the keys from the dictionary `favorite_languages` and assign them one at a time to the variable `name`. The output shows the names of everyone who took the poll:

Jen
Sarah
Edward
Phil

El bucle a través de las claves es en realidad el comportamiento por defecto cuando se hace un bucle a través de un diccionario, por lo que este código tendría exactamente el mismo resultado si escribieras:

Looping through the keys is actually the default behavior when looping through a dictionary, so this code would have exactly the same output if you wrote:

`for name in favorite_languages:`

en lugar de:

rather than:

`for name in favorite_languages.keys():`

Puedes elegir utilizar el método `keys()` explícitamente si facilita la lectura de tu código, u omitirlo si lo deseas.

You can choose to use the `keys()` method explicitly if it makes your code easier to read, or you can omit it if you wish.

Puedes acceder al valor asociado a cualquier clave que te interese dentro del bucle, utilizando la clave actual. Imprimamos un mensaje a un par de amigos sobre las lenguas que eligieron. Haremos un bucle a través de los nombres en el diccionario como hicimos anteriormente, pero cuando el nombre coincida con uno de nuestros amigos, mostraremos un mensaje sobre su lengua favorita:

You can access the value associated with any key you care about inside the loop, by using the current key. Let's print a message to a couple of friends about the languages they chose.

We'll loop through the names in the dictionary as we did previously, but when the name matches one of our friends, we'll display a message about their favorite language:

```
favorite_languages = {
    --snip--
}

friends = ['phil', 'sarah']
for name in favorite_languages.keys():
    print(f"Hi {name.title()}")

    ❶ if name in friends:
    ❷     language = favorite_languages[name].title()
        print(f"\t{name.title()}, I see you love {language}!")
```

En primer lugar, hacemos una lista de amigos a los que queremos imprimir un mensaje. Dentro del bucle, imprimimos el nombre de cada persona. Luego comprobamos si el `name` con el que estamos trabajando está en la lista `friends` ❶. Si lo está, determinamos el idioma favorito de la persona utilizando el nombre del diccionario y el valor actual de `name` como clave ❷. A continuación, imprimimos un saludo especial, que incluye una referencia a la lengua elegida.

First, we make a list of friends that we want to print a message to. Inside the loop, we print each person's name. Then we check whether the `name` we're working with is in the list `friends` ❶. If it is, we determine the person's favorite language using the name of the dictionary and the current value of `name` as the key ❷. We then print a special greeting, including a reference to their language of choice.

Se imprime el nombre de todos, pero nuestros amigos reciben un mensaje especial:

Everyone's name is printed, but our friends receive a special message:

```
Hi Jen.
Hi Sarah.
    Sarah, I see you love C!
Hi Edward.
```

Hi Phil.

Phil, I see you love Python!

También puedes utilizar el método `keys()` para averiguar si se ha encuestado a una persona concreta. Esta vez, vamos a averiguar si Erin participó en la encuesta:

You can also use the `keys()` method to find out if a particular person was polled. This time, let's find out if Erin took the poll:

```
favorite_languages = {  
    --snip--  
}
```

```
if 'erin' not in favorite_languages.keys():  
    print("Erin, please take our poll!")
```

El método `keys()` no es sólo para hacer bucles: en realidad devuelve una secuencia de todas las claves, y la sentencia `if` simplemente comprueba si 'erin' está en esta secuencia. Como no lo está, se imprime un mensaje invitándola a hacer la encuesta:

The `keys()` method isn't just for looping: it actually returns a sequence of all the keys, and the `if` statement simply checks if 'erin' is in this sequence. Because she's not, a message is printed inviting her to take the poll:

Erin, please take our poll!

Recorrer las claves de un diccionario en un orden determinado (*Looping Through a Dictionary's Keys in a Particular Order*)

Recorrer un diccionario en bucle devuelve los elementos en el mismo orden en que se insertaron. Sin embargo, a veces querrás recorrer un diccionario en un orden diferente.

Looping through a dictionary returns the items in the same order they were inserted. Sometimes, though, you'll want to loop through a dictionary in a different order.

Una forma de hacerlo es ordenar las claves a medida que se devuelven en el bucle `for`. Puedes utilizar la función `sorted()` para obtener una copia de las claves en orden:

One way to do this is to sort the keys as they're returned in the `for` loop. You can use the `sorted()` function to get a copy of the keys in order:

```
favorite_languages = {
    'jen': 'python',
    'sarah': 'c',
    'edward': 'rust',
    'phil': 'python',
}
```

```
for name in sorted(favorite_languages.keys()):
    print(f'{name.title()}, thank you for taking the poll.')
```

Esta sentencia `for` es como otras sentencias `for`, salvo que hemos envuelto la función `sorted()` alrededor del método `dictionary.keys()`. Esto le dice a Python que obtenga todas las claves del diccionario y las ordene antes de iniciar el bucle. La salida muestra a todos los que han participado en la encuesta, con los nombres ordenados:

This `for` statement is like other `for` statements, except that we've wrapped the `sorted()` function around the `dictionary.keys()` method. This tells Python to get all the keys in the dictionary and sort them before starting the loop. The output shows everyone who took the poll, with the names displayed in order:

```
Edward, thank you for taking the poll.
Jen, thank you for taking the poll.
Phil, thank you for taking the poll.
Sarah, thank you for taking the poll.
```

Recorrer en bucle todos los valores de un diccionario (*Looping Through All Values in a Dictionary*)

Si lo que te interesa principalmente son los valores que contiene un diccionario, puedes utilizar el método `values()` para devolver una secuencia de valores sin ninguna clave. Por ejemplo, digamos que simplemente queremos una lista de todos los idiomas elegidos en nuestra encuesta sobre lenguajes de programación, sin el nombre de la persona que eligió cada idioma:

If you are primarily interested in the values that a dictionary contains, you can use the `values()` method to return a sequence of values without any keys. For example, say we simply want a list of all languages chosen in our programming language poll, without the name of the person who chose each language:

```
favorite_languages = {
    'jen': 'python',
    'sarah': 'c',
    'edward': 'rust',
    'phil': 'python',
}

print("The following languages have been mentioned:")
for language in favorite_languages.values():
    print(language.title())
```

La sentencia `for` extrae aquí cada valor del diccionario y lo asigna a la variable `language` . Cuando se imprimen estos valores, obtenemos una lista de todos los lenguajes elegidos:

The `for` statement here pulls each value from the dictionary and assigns it to the variable `language` . When these values are printed, we get a list of all chosen languages:

The following languages have been mentioned:

```
Python
C
Rust
Python
```

Este método extrae todos los valores del diccionario sin comprobar si hay repeticiones. Esto podría funcionar bien con un número pequeño de valores, pero en una encuesta con un gran número de encuestados, daría como resultado una lista muy repetitiva. Para ver cada lengua elegida sin repeticiones, podemos utilizar un conjunto. Un *set* es una colección en la que cada elemento debe ser único:

This approach pulls all the values from the dictionary without checking for repeats. This might work fine with a small number of values, but in a poll with a large number of respondents, it would result in a very repetitive list. To see each language chosen without repetition, we can use a set. A *set* is a collection in which each item must be unique:

```
favorite_languages = {  
    --snip--  
}
```

```
print("The following languages have been mentioned:")  
for language in set(favorite_languages.values()):  
    print(language.title())
```

Cuando envuelves `set()` alrededor de una colección de valores que contiene elementos duplicados, Python identifica los elementos únicos de la colección y construye un conjunto a partir de esos elementos. Aquí utilizamos `set()` para extraer los lenguajes únicos de `favorite_languages.values()` .

When you wrap `set()` around a collection of values that contains duplicate items, Python identifies the unique items in the collection and builds a set from those items. Here we use `set()` to pull out the unique languages in `favorite_languages.values()` .

El resultado es una lista no repetitiva de los idiomas que han mencionado las personas que han hecho la encuesta:

The result is a nonrepetitive list of languages that have been mentioned by people taking the poll:

The following languages have been mentioned:

```
Python  
C  
Rust
```

A medida que sigas aprendiendo sobre Python, a menudo encontrarás una función integrada en el lenguaje que te ayudará a hacer exactamente lo que quieres con tus datos.

As you continue learning about Python, you'll often find a built-in feature of the language that helps you do exactly what you want with your data.

Nota (Note)

Puedes construir un conjunto directamente utilizando llaves y separando los elementos con comas:

You can build a set directly using braces and separating the elements with commas:

```
>>> languages = {'python', 'rust', 'python', 'c'}
>>> languages
{'rust', 'python', 'c'}
```

Es fácil confundir los conjuntos con los diccionarios porque ambos están envueltos en llaves. Cuando veas llaves pero no pares clave-valor, probablemente estés ante un conjunto. A diferencia de las listas y los diccionarios, los conjuntos no conservan los elementos en ningún orden específico.

It's easy to mistake sets for dictionaries because they're both wrapped in braces. When you see braces but no key-value pairs, you're probably looking at a set. Unlike lists and dictionaries, sets do not retain items in any specific order.

Pruébalo tú mismo (Try It Yourself)

6-4. Glosario 2: Ahora que sabes cómo hacer un bucle a través de un diccionario, limpia el código del Ejercicio 6-3 (página 99) sustituyendo tu serie de llamadas a `print()` por un bucle que recorra las claves y valores del diccionario. Cuando estés seguro de que tu bucle funciona, añade cinco términos Python más a tu glosario. Cuando vuelvas a ejecutar tu programa, estas nuevas palabras y significados deberían incluirse automáticamente en la salida.

6-4. Glossary 2: Now that you know how to loop through a dictionary, clean up the code from Exercise 6-3 (page 99) by replacing your series of `print()` calls with a loop that runs through the dictionary's keys and values. When you're sure that your loop works, add

five more Python terms to your glossary. When you run your program again, these new words and meanings should automatically be included in the output.

6-5. Ríos: Haz un diccionario que contenga tres ríos principales y el país por el que pasa cada río. Un par clave-valor podría ser `'nile': 'egypt'` .

6-5. Rivers: Make a dictionary containing three major rivers and the country each river runs through. One key-value pair might be `'nile': 'egypt'` .

Listado en español:

- Utiliza un bucle para imprimir una frase sobre cada río, como *The Nile runs through Egypt* .
- Utiliza un bucle para imprimir el nombre de cada río incluido en el diccionario.
- Utiliza un bucle para imprimir el nombre de cada país incluido en el diccionario.

Listing in English:

- Use a loop to print a sentence about each river, such as *The Nile runs through Egypt* .
- Use a loop to print the name of each river included in the dictionary.
- Use a loop to print the name of each country included in the dictionary.

6-6. Sondeo: Utiliza el código de *favorite_languages.py* (página 96).

6-6. Polling: Use the code in *favorite_languages.py* (page 96).

Listado en español:

- Haz una lista de las personas que deberían participar en la encuesta de lenguas favoritas. Incluye algunos nombres que ya estén en el diccionario y otros que no.
- Recorre la lista de personas que deberían participar en la encuesta. Si ya han participado en la encuesta, imprime un mensaje dándoles las gracias por responder. Si aún no han respondido, imprime un mensaje invitándoles a hacerlo.

Listing in English:

- Make a list of people who should take the favorite languages poll. Include some names that are already in the dictionary and some that are not.

- Loop through the list of people who should take the poll. If they have already taken the poll, print a message thanking them for responding. If they have not yet taken the poll, print a message inviting them to take the poll.
-

Anidar (Nesting)

A veces querrás almacenar varios diccionarios en una lista, o una lista de elementos como valor en un diccionario. Esto se llama *nesting*. Puedes anidar diccionarios dentro de una lista, una lista de elementos dentro de un diccionario, o incluso un diccionario dentro de otro diccionario. El anidamiento es una función muy potente, como demostrarán los siguientes ejemplos.

Sometimes you'll want to store multiple dictionaries in a list, or a list of items as a value in a dictionary. This is called *nesting*. You can nest dictionaries inside a list, a list of items inside a dictionary, or even a dictionary inside another dictionary. Nesting is a powerful feature, as the following examples will demonstrate.

Una lista de diccionarios (A List of Dictionaries)

El diccionario `alien_0` contiene información variada sobre un alienígena, pero no tiene espacio para almacenar información sobre un segundo alienígena, y mucho menos sobre una pantalla llena de alienígenas. ¿Cómo puedes gestionar una flota de alienígenas? Una forma es hacer una lista de alienígenas en la que cada alienígena sea un diccionario de información sobre ese alienígena. Por ejemplo, el siguiente código construye una lista de tres alienígenas:

The `alien_0` dictionary contains a variety of information about one alien, but it has no room to store information about a second alien, much less a screen full of aliens. How can you manage a fleet of aliens? One way is to make a list of aliens in which each alien is a dictionary of information about that alien. For example, the following code builds a list of three aliens:

aliens.py

aliens.py

```
alien_0 = {'color': 'green', 'points': 5}
alien_1 = {'color': 'yellow', 'points': 10}
alien_2 = {'color': 'red', 'points': 15}
```

```
❶ aliens = [alien_0, alien_1, alien_2]
```

```
for alien in aliens:
    print(alien)
```

Primero creamos tres diccionarios, cada uno de los cuales representa a un alienígena distinto. Almacenamos cada uno de estos diccionarios en una lista llamada `aliens` ❶. Por último, recorremos la lista e imprimimos cada alienígena:

We first create three dictionaries, each representing a different alien. We store each of these dictionaries in a list called `aliens` ❶. Finally, we loop through the list and print out each alien:

```
{'color': 'green', 'points': 5}
{'color': 'yellow', 'points': 10}
{'color': 'red', 'points': 15}
```

Un ejemplo más realista implicaría más de tres alienígenas con un código que generara automáticamente cada alienígena. En el siguiente ejemplo, utilizamos `range()` para crear una flota de 30 alienígenas:

A more realistic example would involve more than three aliens with code that automatically generates each alien. In the following example, we use `range()` to create a fleet of 30 aliens:

```
# Make an empty list for storing aliens.
aliens = []
```

```
# Make 30 green aliens.
```

```
❶ for alien_number in range(30):
❷     new_alien = {'color': 'green', 'points': 5, 'speed': 'slow'}
❸     aliens.append(new_alien)
```

```
# Show the first 5 aliens.
```

```
❹ for alien in aliens[:5]:
```

```
print(alien)
print("...")

# Show how many aliens have been created.
print(f"Total number of aliens: {len(aliens)}")
```

Este ejemplo comienza con una lista vacía para contener todos los alienígenas que se crearán. La función `range()` ❶ devuelve una serie de números, que sólo indican a Python cuántas veces queremos que se repita el bucle. Cada vez que se ejecuta el bucle, creamos un nuevo alienígena ❷ y añadimos cada nuevo alienígena a la lista `aliens` ❸. Utilizamos una rebanada para imprimir los cinco primeros alienígenas ❹, y finalmente, imprimimos la longitud de la lista para demostrar que realmente hemos generado la flota completa de 30 alienígenas:

This example begins with an empty list to hold all of the aliens that will be created. The `range()` function ❶ returns a series of numbers, which just tells Python how many times we want the loop to repeat. Each time the loop runs, we create a new alien ❷ and then append each new alien to the list `aliens` ❸. We use a slice to print the first five aliens ❹, and finally, we print the length of the list to prove we've actually generated the full fleet of 30 aliens:

```
{'color': 'green', 'points': 5, 'speed': 'slow'}
{'color': 'green', 'points': 5, 'speed': 'slow'}
{'color': 'green', 'points': 5, 'speed': 'slow'}
{'color': 'green', 'points': 5, 'speed': 'slow'}
{'color': 'green', 'points': 5, 'speed': 'slow'}
...
```

Total number of aliens: 30

Todos estos alienígenas tienen las mismas características, pero Python considera a cada uno un objeto independiente, lo que nos permite modificar cada alienígena individualmente.

These aliens all have the same characteristics, but Python considers each one a separate object, which allows us to modify each alien individually.

¿Cómo podrías trabajar con un grupo de alienígenas como éste? Imagina que en un aspecto del juego algunos alienígenas cambian de color y se mueven más rápido a medida que

avanza la partida. Cuando llegue el momento de cambiar los colores, podemos utilizar un bucle `for` y una sentencia `if` para cambiar el color de los alienígenas. Por ejemplo, para cambiar los tres primeros aliens a amarillo, aliens de velocidad media que valen 10 puntos cada uno, podríamos hacer esto:

How might you work with a group of aliens like this? Imagine that one aspect of a game has some aliens changing color and moving faster as the game progresses. When it's time to change colors, we can use a `for` loop and an `if` statement to change the color of the aliens. For example, to change the first three aliens to yellow, medium-speed aliens worth 10 points each, we could do this:

```
# Make an empty list for storing aliens.
aliens = []

# Make 30 green aliens.
for alien_number in range(30):
    new_alien = {'color': 'green', 'points': 5, 'speed': 'slow'}
    aliens.append(new_alien)

for alien in aliens[:3]:
    if alien['color'] == 'green':
        alien['color'] = 'yellow'
        alien['speed'] = 'medium'
        alien['points'] = 10

# Show the first 5 aliens.
for alien in aliens[:5]:
    print(alien)
print("...")
```

Como queremos modificar los tres primeros alienígenas, hacemos un bucle a través de una porción que incluya sólo a los tres primeros alienígenas. Ahora todos los alienígenas son verdes, pero no siempre será así, así que escribimos una sentencia `if` para asegurarnos de que sólo modificamos los alienígenas verdes. Si el alienígena es verde, cambiamos el color a 'yellow', la velocidad a 'medium', y el valor en puntos a 10, como se muestra en la siguiente salida:

Because we want to modify the first three aliens, we loop through a slice that includes only the first three aliens. All of the aliens are green now, but that won't always be the case, so we write an `if` statement to make sure we're only modifying green aliens. If the alien is

green, we change the color to 'yellow' , the speed to 'medium' , and the point value to 10 , as shown in the following output:

```
{'color': 'yellow', 'points': 10, 'speed': 'medium'}
{'color': 'yellow', 'points': 10, 'speed': 'medium'}
{'color': 'yellow', 'points': 10, 'speed': 'medium'}
{'color': 'green', 'points': 5, 'speed': 'slow'}
{'color': 'green', 'points': 5, 'speed': 'slow'}
...
```

Podrías ampliar este bucle añadiendo un bloque `elif` que convierta a los alienígenas amarillos en rojos y rápidos, con un valor de 15 puntos cada uno. Sin mostrar de nuevo todo el programa, ese bucle tendría el siguiente aspecto:

You could expand this loop by adding an `elif` block that turns yellow aliens into red, fast-moving ones worth 15 points each. Without showing the entire program again, that loop would look like this:

```
for alien in aliens[0:3]:
    if alien['color'] == 'green':
        alien['color'] = 'yellow'
        alien['speed'] = 'medium'
        alien['points'] = 10
    elif alien['color'] == 'yellow':
        alien['color'] = 'red'
        alien['speed'] = 'fast'
        alien['points'] = 15
```

Es habitual almacenar varios diccionarios en una lista cuando cada diccionario contiene muchos tipos de información sobre un objeto. Por ejemplo, podrías crear un diccionario para cada usuario de un sitio web, como hicimos en *user.py* en la página 99 , y almacenar los diccionarios individuales en una lista llamada `users` . Todos los diccionarios de la lista deben tener una estructura idéntica, para que puedas recorrer la lista y trabajar con cada objeto diccionario de la misma manera.

It's common to store a number of dictionaries in a list when each dictionary contains many kinds of information about one object. For example, you might create a dictionary for each user on a website, as we did in *user.py* on page 99 , and store the individual dictionaries

in a list called `users` . All of the dictionaries in the list should have an identical structure, so you can loop through the list and work with each dictionary object in the same way.

Una lista dentro de un diccionario (*A List in a Dictionary*)

En lugar de poner un diccionario dentro de una lista, a veces es útil poner una lista dentro de un diccionario. Por ejemplo, considera cómo podrías describir una pizza que alguien está pidiendo. Si sólo utilizaras una lista, lo único que podrías almacenar sería una lista de los ingredientes de la pizza. Con un diccionario, una lista de ingredientes puede ser sólo un aspecto de la pizza que estás describiendo.

Rather than putting a dictionary inside a list, it's sometimes useful to put a list inside a dictionary. For example, consider how you might describe a pizza that someone is ordering. If you were to use only a list, all you could really store is a list of the pizza's toppings. With a dictionary, a list of toppings can be just one aspect of the pizza you're describing.

En el siguiente ejemplo, se almacenan dos tipos de información para cada pizza: un tipo de masa y una lista de ingredientes. La lista de ingredientes es un valor asociado a la clave `'toppings'` . Para utilizar los elementos de la lista, damos el nombre del diccionario y la clave `'toppings'` , como haríamos con cualquier valor del diccionario. En lugar de devolver un único valor, obtenemos una lista de ingredientes:

In the following example, two kinds of information are stored for each pizza: a type of crust and a list of toppings. The list of toppings is a value associated with the key `'toppings'` . To use the items in the list, we give the name of the dictionary and the key `'toppings'` , as we would any value in the dictionary. Instead of returning a single value, we get a list of toppings:

pizza.py

pizza.py

```
# Store information about a pizza being ordered.
pizza = {
    'crust': 'thick',
```

```
'toppings': ['mushrooms', 'extra cheese'],  
}
```

```
# Summarize the order.
```

```
❶ print(f"You ordered a {pizza['crust']}-crust pizza "  
    "with the following toppings:")
```

```
❷ for topping in pizza['toppings']:  
    print(f"\t{topping}")
```

Empezamos con un diccionario que contiene información sobre una pizza que se ha pedido. Una de las claves del diccionario es 'crust' , y el valor asociado es la cadena 'thick' . La siguiente clave, 'toppings' , tiene como valor una lista que almacena todos los ingredientes solicitados. Resumimos el pedido antes de construir la pizza ❶ . Cuando necesites dividir una línea larga en una llamada a `print()` , elige un punto apropiado en el que dividir la línea que se está imprimiendo, y termina la línea con una comilla. Pon sangría en la línea siguiente, añade una comilla de apertura y continúa la cadena. Python combinará automáticamente todas las cadenas que encuentre dentro de los paréntesis. Para imprimir los ingredientes, escribimos un bucle `for` ❷ . Para acceder a la lista de ingredientes, utilizamos la clave 'toppings' , y Python obtiene la lista de ingredientes del diccionario.

We begin with a dictionary that holds information about a pizza that has been ordered. One key in the dictionary is 'crust' , and the associated value is the string 'thick' . The next key, 'toppings' , has a list as its value that stores all requested toppings. We summarize the order before building the pizza ❶ . When you need to break up a long line in a `print()` call, choose an appropriate point at which to break the line being printed, and end the line with a quotation mark. Indent the next line, add an opening quotation mark, and continue the string. Python will automatically combine all of the strings it finds inside the parentheses. To print the toppings, we write a `for` loop ❷ . To access the list of toppings, we use the key 'toppings' , and Python grabs the list of toppings from the dictionary.

La siguiente salida resume la pizza que pensamos construir:

The following output summarizes the pizza that we plan to build:

```
You ordered a thick-crust pizza with the following toppings:  
mushrooms  
extra cheese
```

Puedes anidar una lista dentro de un diccionario siempre que quieras asociar más de un valor a una sola clave de un diccionario. En el ejemplo anterior de los lenguajes de programación favoritos, si almacenáramos las respuestas de cada persona en una lista, la gente podría elegir más de un lenguaje favorito. Cuando hagamos un bucle en el diccionario, el valor asociado a cada persona sería una lista de idiomas en lugar de un único idioma. Dentro del bucle `for` del diccionario, utilizamos otro bucle `for` para recorrer la lista de lenguas asociadas a cada persona:

You can nest a list inside a dictionary anytime you want more than one value to be associated with a single key in a dictionary. In the earlier example of favorite programming languages, if we were to store each person's responses in a list, people could choose more than one favorite language. When we loop through the dictionary, the value associated with each person would be a list of languages rather than a single language. Inside the dictionary's `for` loop, we use another `for` loop to run through the list of languages associated with each person:

idiomas_favoritos.py

favorite_languages.py

```
favorite_languages = {
    'jen': ['python', 'rust'],
    'sarah': ['c'],
    'edward': ['rust', 'go'],
    'phil': ['python', 'haskell'],
}
```

```
❶ for name, languages in favorite_languages.items():
    print(f"\n{name.title()}s favorite languages are:")
    ❷ for language in languages:
        print(f"\t{language.title()}")
```

El valor asociado a cada nombre en `favorite_languages` es ahora una lista. Observa que algunas personas tienen un idioma favorito y otras tienen varios favoritos. Cuando hacemos un bucle a través del diccionario ❶, utilizamos el nombre de variable `languages` para guardar cada valor del diccionario, porque sabemos que cada valor será una lista. Dentro del bucle principal del diccionario, utilizamos otro bucle `for` ❷ para recorrer la lista de lenguas favoritas de cada persona. Ahora cada persona puede enumerar tantas lenguas favoritas como quiera:

The value associated with each name in `favorite_languages` is now a list. Note that some people have one favorite language and others have multiple favorites. When we loop through the dictionary ❶, we use the variable name `languages` to hold each value from the dictionary, because we know that each value will be a list. Inside the main dictionary loop, we use another `for` loop ❷ to run through each person's list of favorite languages. Now each person can list as many favorite languages as they like:

Jen's favorite languages are:

Python
Rust

Sarah's favorite languages are:

C

Edward's favorite languages are:

Rust
Go

Phil's favorite languages are:

Python
Haskell

Para refinar aún más este programa, podrías incluir una sentencia `if` al principio del bucle `for` del diccionario para ver si cada persona tiene más de una lengua favorita examinando el valor de `len(languages)`. Si una persona tiene más de una favorita, la salida seguiría siendo la misma. Si la persona sólo tiene una lengua favorita, podrías cambiar la redacción para reflejarlo. Por ejemplo, podrías decir: "El lenguaje favorito de Sara es C".

To refine this program even further, you could include an `if` statement at the beginning of the dictionary's `for` loop to see whether each person has more than one favorite language by examining the value of `len(languages)`. If a person has more than one favorite, the output would stay the same. If the person has only one favorite language, you could change the wording to reflect that. For example, you could say, "Sarah's favorite language is C."

Nota (Note)

No debes anidar listas y diccionarios demasiado profundamente. Si estás anidando elementos mucho más profundamente de lo que se ve en los ejemplos anteriores, o si estás trabajando con el código de otra persona con niveles significativos de anidamiento, lo más probable es que haya una forma más sencilla de resolver el problema.

You should not nest lists and dictionaries too deeply. If you're nesting items much deeper than what you see in the preceding examples, or if you're working with someone else's code with significant levels of nesting, there's most likely a simpler way to solve the problem.

Un diccionario dentro de otro diccionario (*A Dictionary in a Dictionary*)

Puedes anidar un diccionario dentro de otro diccionario, pero tu código puede complicarse rápidamente cuando lo hagas. Por ejemplo, si tienes varios usuarios para un sitio web, cada uno con un nombre de usuario único, puedes utilizar los nombres de usuario como claves de un diccionario. A continuación, puedes almacenar información sobre cada usuario utilizando un diccionario como valor asociado a su nombre de usuario. En el siguiente listado, almacenamos tres datos sobre cada usuario: su nombre, apellidos y ubicación. Accederemos a esta información recorriendo los nombres de usuario y el diccionario de información asociado a cada nombre de usuario:

You can nest a dictionary inside another dictionary, but your code can get complicated quickly when you do. For example, if you have several users for a website, each with a unique username, you can use the usernames as the keys in a dictionary. You can then store information about each user by using a dictionary as the value associated with their username. In the following listing, we store three pieces of information about each user: their first name, last name, and location. We'll access this information by looping through the usernames and the dictionary of information associated with each username:

`muchos_usuarios.py`

`many_users.py`

```

users = {
    'aeinstein': {
        'first': 'albert',
        'last': 'einstein',
        'location': 'princeton',
    },

    'mcurie': {
        'first': 'marie',
        'last': 'curie',
        'location': 'paris',
    },
}

```

```

❶ for username, user_info in users.items():
❷     print(f"\nUsername: {username}")
❸     full_name = f'{user_info['first']} {user_info['last']}'
        location = user_info['location']

❹     print(f"\tFull name: {full_name.title()}")
        print(f"\tLocation: {location.title()}")

```

Primero definimos un diccionario llamado `users` con dos claves: una para cada uno de los nombres de usuario `'aeinstein'` y `'mcurie'`. El valor asociado a cada clave es un diccionario que incluye el nombre, apellidos y ubicación de cada usuario. A continuación, recorreremos el diccionario `users` ❶. Python asigna cada clave a la variable `username`, y el diccionario asociado a cada nombre de usuario se asigna a la variable `user_info`. Una vez dentro del bucle principal del diccionario, imprimimos el nombre de usuario ❷.

We first define a dictionary called `users` with two keys: one each for the usernames `'aeinstein'` and `'mcurie'`. The value associated with each key is a dictionary that includes each user's first name, last name, and location. Then, we loop through the `users` dictionary ❶. Python assigns each key to the variable `username`, and the dictionary associated with each username is assigned to the variable `user_info`. Once inside the main dictionary loop, we print the username ❷.

A continuación, empezamos a acceder al diccionario interno ❸. La variable `user_info`, que contiene el diccionario de información del usuario, tiene tres claves: `'first'`, `'last'`, y `'location'`. Utilizamos cada clave para generar un nombre completo y una ubicación de cada persona con un formato ordenado y, a continuación, imprimimos un resumen de lo

que sabemos de cada usuario ④ :

Then, we start accessing the inner dictionary ③ . The variable `user_info` , which contains the dictionary of user information, has three keys: 'first' , 'last' , and 'location' . We use each key to generate a neatly formatted full name and location for each person, and then print a summary of what we know about each user ④ :

```
Username: aeinstein  
Full name: Albert Einstein  
Location: Princeton
```

```
Username: mcurie  
Full name: Marie Curie  
Location: Paris
```

Observa que la estructura del diccionario de cada usuario es idéntica. Aunque Python no lo requiere, esta estructura facilita el trabajo con diccionarios anidados. Si el diccionario de cada usuario tuviera claves diferentes, el código dentro del bucle `for` sería más complicado.

Notice that the structure of each user's dictionary is identical. Although not required by Python, this structure makes nested dictionaries easier to work with. If each user's dictionary had different keys, the code inside the `for` loop would be more complicated.

Pruébalo tú mismo (*Try It Yourself*)

6-7. Personas: Empieza con el programa que escribiste para el Ejercicio 6-1 (página 98). Crea dos nuevos diccionarios que representen a personas diferentes, y almacena los tres diccionarios en una lista llamada `people` . Recorre tu lista de personas. Mientras recorres la lista, imprime todo lo que sepas sobre cada persona.

6-7. People: Start with the program you wrote for Exercise 6-1 (page 98). Make two new dictionaries representing different people, and store all three dictionaries in a list called `people` . Loop through your list of people. As you loop through the list, print everything you know about each person.

6-8. Mascotas: Haz varios diccionarios, donde cada diccionario represente una mascota diferente. En cada diccionario, incluye el tipo de animal y el nombre del dueño. Guarda estos diccionarios en una lista llamada `pets` . A continuación, recorre la lista en bucle y, mientras lo haces, imprime todo lo que sepas sobre cada mascota.

6-8. Pets: Make several dictionaries, where each dictionary represents a different pet. In each dictionary, include the kind of animal and the owner's name. Store these dictionaries in a list called `pets` . Next, loop through your list and as you do, print everything you know about each pet.

6-9. Lugares favoritos: Crea un diccionario llamado `favorite_places` . Piensa en tres nombres para utilizarlos como claves en el diccionario, y almacena de uno a tres lugares favoritos para cada persona. Para hacer este ejercicio un poco más interesante, pide a algunos amigos que nombren algunos de sus lugares favoritos. Recorre el diccionario e imprime el nombre de cada persona y sus lugares favoritos.

6-9. Favorite Places: Make a dictionary called `favorite_places` . Think of three names to use as keys in the dictionary, and store one to three favorite places for each person. To make this exercise a bit more interesting, ask some friends to name a few of their favorite places. Loop through the dictionary, and print each person's name and their favorite places.

6-10. Números Favoritos: Modifica tu programa del Ejercicio 6-2 (página 98) para que cada persona pueda tener más de un número favorito. Luego imprime el nombre de cada persona junto con sus números favoritos.

6-10. Favorite Numbers: Modify your program from Exercise 6-2 (page 98) so each person can have more than one favorite number. Then print each person's name along with their favorite numbers.

6-11. Ciudades: Crea un diccionario llamado `cities` . Utiliza los nombres de tres ciudades como claves de tu diccionario. Crea un diccionario de información sobre cada ciudad e incluye el país en el que se encuentra la ciudad, su población aproximada y un dato sobre esa ciudad. Las claves del diccionario de cada ciudad deben ser algo así como `country` , `population` , y `fact` . Imprime el nombre de cada ciudad y toda la información que hayas

almacenado sobre ella.

6-11. Cities: Make a dictionary called `cities` . Use the names of three cities as keys in your dictionary. Create a dictionary of information about each city and include the country that the city is in, its approximate population, and one fact about that city. The keys for each city's dictionary should be something like `country` , `population` , and `fact` . Print the name of each city and all of the information you have stored about it.

6-12. Ampliaciones: Ahora estamos trabajando con ejemplos lo suficientemente complejos como para que puedan ampliarse de muchas maneras. Utiliza uno de los programas de ejemplo de este capítulo y amplíalo añadiendo nuevas claves y valores, cambiando el contexto del programa o mejorando el formato de la salida.

6-12. Extensions: We're now working with examples that are complex enough that they can be extended in any number of ways. Use one of the example programs from this chapter, and extend it by adding new keys and values, changing the context of the program, or improving the formatting of the output.

Resumen (Summary)

En este capítulo has aprendido a definir un diccionario y a trabajar con la información almacenada en él. Aprendiste a acceder y modificar elementos individuales de un diccionario, y a recorrer en bucle todos los de la información de un diccionario. Aprendiste a recorrer los pares clave-valor de un diccionario, sus claves y sus valores. También has aprendido a anidar varios diccionarios en una lista, a anidar listas en un diccionario y a anidar un diccionario dentro de otro diccionario.

In this chapter, you learned how to define a dictionary and how to work with the information stored in a dictionary. You learned how to access and modify individual elements in a dictionary, and how to loop through all of the information in a dictionary. You learned to loop through a dictionary's key-value pairs, its keys, and its values. You also learned how to nest multiple dictionaries in a list, nest lists in a dictionary, and nest a dictionary inside a dictionary.

En el próximo capítulo aprenderás sobre los bucles `while` y cómo aceptar entradas de personas que utilizan tus programas. Éste será un capítulo emocionante, porque aprenderás a hacer que todos tus programas sean interactivos: podrán responder a las entradas del usuario.

In the next chapter you'll learn about `while` loops and how to accept input from people who are using your programs. This will be an exciting chapter, because you'll learn to make all of your programs interactive: they'll be able to respond to user input.

Capítulo 7: Entrada del usuario y bucles while (Chapter 7: User Input and while Loops)

7

Entrada de usuario y bucles while (7 User Input and while Loops)



DESCRIPCIÓN DE LA
IMAGEN (En español):

La imagen muestra a una serpiente en blanco y negro rodeando un cohete. La serpiente tiene una expresión decidida, con cejas arqueadas, y parece estar avanzando rápidamente. El cohete, con un diseño estilizado y puntiagudo, está orientado hacia el frente, simbolizando velocidad y dinamismo. El fondo es de un color gris claro.

La mayoría de los programas se escriben para resolver un problema de un usuario final. Para ello, normalmente necesitas obtener alguna información del usuario. Por ejemplo, supongamos que alguien quiere averiguar si tiene edad suficiente para votar. Si escribes un programa para responder a esta pregunta, necesitarás saber la edad del usuario antes de poder darle una respuesta. El programa tendrá que pedir al usuario que introduzca, o *input*, su edad; una vez que el programa tenga esta información, podrá compararla con la edad para votar para determinar si el usuario tiene la edad suficiente e informar del resultado.

Most programs are written to solve an end user's problem. To do so, you usually need to get some information from the user. For example, say someone wants to find out whether they're old enough to vote. If you write a program to answer this question, you need to know the user's age before you can provide an answer. The program will need to ask the user to enter, or *input*, their age; once the program has this input, it can compare it to the voting age to determine if the user is old enough and then report the result.

En este capítulo aprenderás a aceptar las entradas del usuario para que tu programa pueda trabajar con ellas. Cuando tu programa necesite un nombre, podrás pedirselo al usuario. Cuando tu programa necesite una lista de nombres, podrás pedir al usuario una serie de nombres. Para ello, utilizarás la función `input()` .

In this chapter you'll learn how to accept user input so your program can then work with it. When your program needs a name, you'll be able to prompt the user for a name. When your program needs a list of names, you'll be able to prompt the user for a series of names. To do this, you'll use the `input()` function.

También aprenderás a mantener los programas en ejecución todo el tiempo que los usuarios deseen, para que puedan introducir toda la información que necesiten; después, tu programa podrá trabajar con esa información. Utilizarás el bucle `while` de Python para mantener los programas en ejecución mientras se cumplan determinadas condiciones.

You'll also learn how to keep programs running as long as users want them to, so they can enter as much information as they need to; then, your program can work with that information. You'll use Python's `while` loop to keep programs running as long as certain conditions remain true.

Con la capacidad de trabajar con la información introducida por el usuario y la posibilidad de controlar cuánto tiempo se ejecutan tus programas, podrás escribir programas totalmente interactivos.

With the ability to work with user input and the ability to control how long your programs run, you'll be able to write fully interactive programs.

Cómo funciona la función `input()` *(How the `input()` Function Works)*

La función `input()` pausa tu programa y espera a que el usuario introduzca algún texto. Una vez que Python recibe la entrada del usuario, la asigna a una variable para que te resulte más cómodo trabajar con ella.

The `input()` function pauses your program and waits for the user to enter some text. Once Python receives the user's input, it assigns that input to a variable to make it convenient for you to work with.

Por ejemplo, el siguiente programa pide al usuario que introduzca un texto y, a continuación, le devuelve el mensaje:

For example, the following program asks the user to enter some text, then displays that message back to the user:

loro.py

parrot.py

```
message = input("Tell me something, and I will repeat it back to you: ")
print(message)
```

La función `input()` toma un argumento: el *prompt* que queremos mostrar al usuario, para que sepa qué tipo de información debe introducir. En este ejemplo, cuando Python ejecuta la primera línea, el usuario ve el prompt `Tell me something, and I will repeat it back to you:` . El programa espera mientras el usuario introduce su respuesta y continúa después de que el usuario pulse ENTER. La respuesta se asigna a la variable `message`, y a continuación `print(message)` muestra la información introducida al usuario:

The `input()` function takes one argument: the *prompt* that we want to display to the user, so they know what kind of information to enter. In this example, when Python runs the first line, the user sees the prompt `Tell me something, and I will repeat it back to you:` . The program waits while the user enters their response and continues after the user presses ENTER. The response is assigned to the variable `message`, then `print(message)` displays the input back to the user:

```
Tell me something, and I will repeat it back to you: Hello everyone!
Hello everyone!
```

Nota (Note)

Algunos editores de texto no ejecutan programas que solicitan la entrada de datos al usuario. Puedes utilizar estos editores para escribir programas que soliciten entradas, pero tendrás que ejecutar estos programas desde un terminal. Consulta " Ejecutar programas Python desde un terminal " en la página 11 .

Some text editors won't run programs that prompt the user for input. You can use these editors to write programs that prompt for input, but you'll need to run these programs from a terminal. See " Running Python Programs from a Terminal " on page 11 .

Escribir instrucciones claras (Writing Clear Prompts)

Cada vez que utilices la función `input()` , debes incluir una instrucción clara y fácil de seguir que indique al usuario exactamente qué tipo de información estás buscando. Cualquier instrucción que indique al usuario lo que debe introducir debería funcionar. Por ejemplo

Each time you use the `input()` function, you should include a clear, easy-to-follow prompt that tells the user exactly what kind of information you're looking for. Any statement that tells the user what to enter should work. For example:

greeter.py

greeter.py

```
name = input("Please enter your name: ")
print(f"\nHello, {name}!")
```

Añade un espacio al final de tus indicaciones (después de los dos puntos en el ejemplo anterior) para separar la indicación de la respuesta del usuario y para dejarle claro dónde introducir el texto. Por ejemplo:

Add a space at the end of your prompts (after the colon in the preceding example) to separate the prompt from the user's response and to make it clear to your user where to enter their text. For example:

```
Please enter your name: Eric
Hello, Eric!
```

A veces querrás escribir un aviso de más de una línea. Por ejemplo, puede que quieras decirle al usuario por qué le pides que introduzca determinada información. Puedes asignar tu pregunta a una variable y pasar esa variable a la función `input()`. Esto te permite construir tu pregunta en varias líneas, y luego escribir una declaración limpia en `input()`.

Sometimes you'll want to write a prompt that's longer than one line. For example, you might want to tell the user why you're asking for certain input. You can assign your prompt to a variable and pass that variable to the `input()` function. This allows you to build your prompt over several lines, then write a clean `input()` statement.

saludador.py

greeter.py

```
prompt = "If you share your name, we can personalize the messages you see."
prompt += "\nWhat is your first name? "
```

```
name = input(prompt)
print(f"\nHello, {name}!")
```

Este ejemplo muestra una forma de construir una cadena de varias líneas. La primera línea asigna la primera parte del mensaje a la variable `prompt`. En la segunda línea, el operador `+=` toma la cadena que se asignó a `prompt` y añade la nueva cadena al final.

This example shows one way to build a multiline string. The first line assigns the first part of the message to the variable `prompt`. In the second line, the operator `+=` takes the string that was assigned to `prompt` and adds the new string onto the end.

El mensaje abarca ahora dos líneas, de nuevo con espacio tras el signo de interrogación para mayor claridad:

The prompt now spans two lines, again with space after the question mark for clarity:

If you share your name, we can personalize the messages you see.
What is your first name? Eric

Hello, Eric!

Uso de `int()` para aceptar entradas numéricas *(Using `int()` to Accept Numerical Input)*

Cuando utilizas la función `input()`, Python interpreta todo lo que introduce el usuario como una cadena. Considera la siguiente sesión del intérprete, que pregunta por la edad del usuario:

When you use the `input()` function, Python interprets everything the user enters as a string. Consider the following interpreter session, which asks for the user's age:

```
>>> age = input("How old are you? ")
How old are you? 21
>>> age
'21'
```

El usuario introduce el número 21, pero cuando pedimos a Python el valor de `age`, nos devuelve `'21'`, la representación en cadena del valor numérico introducido. Sabemos que Python interpretó la entrada como una cadena porque ahora el número está entre comillas. Si lo único que quieres es imprimir la entrada, esto funciona bien. Pero si intentas utilizar la entrada como un número, obtendrás un error:

The user enters the number 21, but when we ask Python for the value of `age`, it returns `'21'`, the string representation of the numerical value entered. We know Python interpreted the input as a string because the number is now enclosed in quotes. If all you want to do is print the input, this works well. But if you try to use the input as a number, you'll get an error:

```
>>> age = input("How old are you? ")
How old are you? 21
```

```
❶ >>> age >= 18
```

```
Traceback (most recent call last):
```

```
File "<stdin>", line 1, in <module>
```

```
❷ TypeError: '>=' not supported between instances of 'str' and 'int'
```

Cuando intentas utilizar la entrada para hacer una comparación numérica ❶ , Python produce un error porque no puede comparar una cadena con un número entero: la cadena '21' que se asigna a `age` no puede compararse con el valor numérico 18 ❷ .

When you try to use the input to do a numerical comparison ❶ , Python produces an error because it can't compare a string to an integer: the string '21' that's assigned to `age` can't be compared to the numerical value 18 ❷ .

Podemos resolver este problema utilizando la función `int()` , que convierte la cadena de entrada en un valor numérico. Esto permite que la comparación se ejecute correctamente:

We can resolve this issue by using the `int()` function, which converts the input string to a numerical value. This allows the comparison to run successfully:

```
>>> age = input("How old are you? ")
How old are you? 21
❶ >>> age = int(age)
>>> age >= 18
True
```

En este ejemplo, cuando introducimos 21 en el indicador, Python interpreta el número como una cadena, pero el valor se convierte en una representación numérica mediante `int()` ❶ . Ahora Python puede ejecutar la prueba condicional: compara `age` (que ahora representa el valor numérico 21) y 18 para ver si `age` es mayor o igual que 18. Esta prueba evalúa a `True` .

In this example, when we enter 21 at the prompt, Python interprets the number as a string, but the value is then converted to a numerical representation by `int()` ❶ . Now Python can run the conditional test: it compares `age` (which now represents the numerical value 21) and 18 to see if `age` is greater than or equal to 18. This test evaluates to `True` .

¿Cómo se utiliza la función `int()` en un programa real? Considera un programa que determina si las personas son lo suficientemente altas para subir a una montaña rusa:

How do you use the `int()` function in an actual program? Consider a program that determines whether people are tall enough to ride a roller coaster:

montaña rusa.py

rollercoaster.py

```
height = input("How tall are you, in inches? ")  
height = int(height)
```

```
if height >= 48:  
    print("\nYou're tall enough to ride!")  
else:  
    print("\nYou'll be able to ride when you're a little older.")
```

El programa puede comparar `height` con 48 porque `height = int(height)` convierte el valor introducido en una representación numérica antes de realizar la comparación. Si el número introducido es mayor o igual que 48, le decimos al usuario que es lo suficientemente alto:

The program can compare `height` to 48 because `height = int(height)` converts the input value to a numerical representation before the comparison is made. If the number entered is greater than or equal to 48, we tell the user that they're tall enough:

How tall are you, in inches? 71

You're tall enough to ride!

Cuando utilices entradas numéricas para hacer cálculos y comparaciones, asegúrate de convertir primero el valor de entrada en una representación numérica.

When you use numerical input to do calculations and comparisons, be sure to convert the input value to a numerical representation first.

El operador módulo (*The Modulo Operator*)

Una herramienta útil para trabajar con información numérica es el *modulo operator* (`%`), que divide un número por otro y devuelve el resto:

A useful tool for working with numerical information is the *modulo operator* (`%`), which divides one number by another number and returns the remainder:

```
>>> 4 % 3
1
>>> 5 % 3
2
>>> 6 % 3
0
>>> 7 % 3
1
```

El operador módulo no te dice cuántas veces cabe un número en otro; sólo te dice cuál es el resto.

The modulo operator doesn't tell you how many times one number fits into another; it only tells you what the remainder is.

Cuando un número es divisible por otro, el resto es 0, por lo que el operador módulo siempre devuelve 0. Puedes utilizar este hecho para determinar si un número es par o impar:

When one number is divisible by another number, the remainder is 0, so the modulo operator always returns 0. You can use this fact to determine if a number is even or odd:

par_o_impar.py

even_or_odd.py

```
number = input("Enter a number, and I'll tell you if it's even or odd: ")
number = int(number)
```

```
if number % 2 == 0:
    print(f"\nThe number {number} is even.")
else:
    print(f"\nThe number {number} is odd.")
```

Los números pares siempre son divisibles por dos, por lo que si el módulo de un número y dos es cero (aquí, `if number % 2 == 0`) el número es par. En caso contrario, es impar.

Even numbers are always divisible by two, so if the modulo of a number and two is zero (here, `if number % 2 == 0`) the number is even. Otherwise, it's odd.

Enter a number, and I'll tell you if it's even or odd: 42

The number 42 is even.

Pruébalo tú mismo (*Try It Yourself*)

7-1. Coche de alquiler: Escribe un programa que pregunte al usuario qué tipo de coche de alquiler le gustaría. Imprime un mensaje sobre ese coche, como "A ver si puedo encontrarte un Subaru".

7-1. Rental Car: Write a program that asks the user what kind of rental car they would like. Print a message about that car, such as "Let me see if I can find you a Subaru."

7-2. Asientos en un restaurante: Escribe un programa que pregunte al usuario cuántas personas hay en su grupo para cenar. Si la respuesta es más de ocho, imprime un mensaje diciendo que tendrán que esperar para una mesa. En caso contrario, informa de que su mesa está lista.

7-2. Restaurant Seating: Write a program that asks the user how many people are in their dinner group. If the answer is more than eight, print a message saying they'll have to wait for a table. Otherwise, report that their table is ready.

7-3. Múltiplos de diez: Pide al usuario un número y, a continuación, informa de si el número es múltiplo de 10 o no.

7-3. Multiples of Ten: Ask the user for a number, and then report whether the number is a multiple of 10 or not.

Introducción a los bucles while *(Introducing while Loops)*

El bucle `for` toma una colección de elementos y ejecuta un bloque de código una vez por cada elemento de la colección. En cambio, el bucle `while` se ejecuta mientras, o *while*, se cumpla una determinada condición.

The `for` loop takes a collection of items and executes a block of code once for each item in the collection. In contrast, the `while` loop runs as long as, or *while*, a certain condition is true.

El bucle while en acción *(The while Loop in Action)*

Puedes utilizar un bucle `while` para contar hacia arriba a través de una serie de números. Por ejemplo, el siguiente bucle `while` cuenta del 1 al 5:

You can use a `while` loop to count up through a series of numbers. For example, the following `while` loop counts from 1 to 5:

contando.py

counting.py

```
current_number = 1
while current_number <= 5:
    print(current_number)
    current_number += 1
```

En la primera línea, empezamos a contar desde 1 asignando a `current_number` el valor 1. A continuación, se establece el bucle `while` para que siga ejecutándose mientras el valor de `current_number` sea menor o igual que 5. El código dentro del bucle imprime el valor de `current_number` y luego añade 1 a ese valor con `current_number += 1`. (El operador `+=` es una abreviatura de `current_number = current_number + 1`.)

In the first line, we start counting from 1 by assigning `current_number` the value 1. The `while` loop is then set to keep running as long as the value of `current_number` is less than or equal to 5. The code inside the loop prints the value of `current_number` and then adds 1 to that value with `current_number += 1`. (The `+=` operator is shorthand for `current_number = current_number + 1`.)

Python repite el bucle mientras la condición `current_number <= 5` sea cierta. Como 1 es menor que 5, Python imprime 1 y luego añade 1, convirtiendo el número actual en 2. Como 2 es menor que 5, Python imprime 2 y vuelve a añadir 1, convirtiendo el número actual en 3, y así sucesivamente. Una vez que el valor de `current_number` es mayor que 5, el bucle deja de ejecutarse y el programa finaliza:

Python repeats the loop as long as the condition `current_number <= 5` is true. Because 1 is less than 5, Python prints 1 and then adds 1, making the current number 2. Because 2 is less than 5, Python prints 2 and adds 1 again, making the current number 3, and so on. Once the value of `current_number` is greater than 5, the loop stops running and the program ends:

```
1
2
3
4
5
```

Lo más probable es que los programas que utilizas a diario contengan bucles `while` . Por ejemplo, un juego necesita un bucle `while` para seguir ejecutándose todo el tiempo que quieras seguir jugando, y así puede dejar de ejecutarse en cuanto le pidas que salga. Los programas no serían divertidos de usar si dejaran de ejecutarse antes de que se lo pidiéramos o siguieran ejecutándose incluso después de que quisiéramos salir, por eso los bucles `while` son bastante útiles.

The programs you use every day most likely contain `while` loops. For example, a game needs a `while` loop to keep running as long as you want to keep playing, and so it can stop running as soon as you ask it to quit. Programs wouldn't be fun to use if they stopped running before we told them to or kept running even after we wanted to quit, so `while` loops are quite useful.

Dejar que el usuario elija cuándo salir (*Letting the User Choose When to Quit*)

Podemos hacer que el programa *parrot.py* se ejecute tanto tiempo como quiera el usuario poniendo la mayor parte del programa dentro de un bucle `while` . Definiremos un *quit value* y luego mantendremos el programa en ejecución mientras el usuario no haya introducido el valor de abandonar:

We can make the *parrot.py* program run as long as the user wants by putting most of the program inside a `while` loop. We'll define a *quit value* and then keep the program running as long as the user has not entered the quit value:

loro.py

parrot.py

```
prompt = "\nTell me something, and I will repeat it back to you:"
prompt += "\nEnter 'quit' to end the program. "
```

```
message = ""
while message != 'quit':
    message = input(prompt)
    print(message)
```

Primero definimos un prompt que indica al usuario sus dos opciones: introducir un mensaje o introducir el valor de abandonar (en este caso, 'quit'). A continuación, creamos una variable `message` para registrar cualquier valor que introduzca el usuario. Definimos `message` como una cadena vacía, `""` , para que Python tenga algo que comprobar la primera vez que llegue a la línea `while` . La primera vez que se ejecuta el programa y Python llega a la sentencia `while` , necesita comparar el valor de `message` con 'quit' , pero todavía no se ha introducido ninguna entrada del usuario. Si Python no tiene nada que comparar, no podrá seguir ejecutando el programa. Para resolver este problema de , nos aseguramos de dar a `message` un valor inicial. Aunque sólo sea una cadena vacía, tendrá sentido para Python y le permitirá realizar la comparación que hace funcionar el bucle `while` . Este bucle `while` se ejecuta mientras el valor de `message` no sea 'quit' .

We first define a prompt that tells the user their two options: entering a message or entering the quit value (in this case, 'quit'). Then we set up a variable `message` to keep track of whatever value the user enters. We define `message` as an empty string, `""` , so Python has something to check the first time it reaches the `while` line. The first time the program runs and Python reaches the `while` statement, it needs to compare the value of `message` to 'quit' , but no user input has been entered yet. If Python has nothing to compare, it won't be able to continue running the program. To solve this problem, we make sure to give `message` an initial value. Although it's just an empty string, it will make sense to Python and allow it to perform the comparison that makes the `while` loop work. This `while` loop runs as long as the value of `message` is not 'quit' .

La primera vez que pasa por el bucle, `message` es sólo una cadena vacía, por lo que Python entra en el bucle. En `message = input(prompt)` , Python muestra el prompt y espera a que el usuario introduzca sus datos. Lo que introduzca se asigna a `message` y se imprime; después, Python reevalúa la condición en la sentencia `while` . Mientras el usuario no introduzca la palabra 'quit' , se vuelve a mostrar el prompt y Python espera a que introduzca más datos. Cuando finalmente el usuario introduce 'quit' , Python deja de ejecutar el bucle `while` y el programa termina:

The first time through the loop, `message` is just an empty string, so Python enters the loop. At `message = input(prompt)` , Python displays the prompt and waits for the user to enter their input. Whatever they enter is assigned to `message` and printed; then, Python reevaluates the condition in the `while` statement. As long as the user has not entered the word 'quit' , the prompt is displayed again and Python waits for more input. When the user finally enters 'quit' , Python stops executing the `while` loop and the program ends:

Tell me something, and I will repeat it back to you:
Enter 'quit' to end the program. Hello everyone!
Hello everyone!

Tell me something, and I will repeat it back to you:
Enter 'quit' to end the program. Hello again.
Hello again.

Tell me something, and I will repeat it back to you:
Enter 'quit' to end the program. quit
quit

Este programa funciona bien, excepto porque imprime la palabra 'quit' como si fuera un mensaje real. Una simple comprobación de `if` lo soluciona:

This program works well, except that it prints the word 'quit' as if it were an actual message. A simple `if` test fixes this:

```
prompt = "\nTell me something, and I will repeat it back to you:"  
prompt += "\nEnter 'quit' to end the program. "
```

```
message = ""  
while message != 'quit':  
    message = input(prompt)
```

```
    if message != 'quit':  
        print(message)
```

Ahora el programa hace una comprobación rápida antes de mostrar el mensaje y sólo imprime el mensaje si no coincide con el valor de abandonar:

Now the program makes a quick check before displaying the message and only prints the message if it does not match the quit value:

Tell me something, and I will repeat it back to you:
Enter 'quit' to end the program. Hello everyone!
Hello everyone!

Tell me something, and I will repeat it back to you:

Enter 'quit' to end the program. Hello again.
Hello again.

Tell me something, and I will repeat it back to you:
Enter 'quit' to end the program. quit

Utilizar un indicador (*Using a Flag*)

En el ejemplo anterior, hicimos que el programa realizara determinadas tareas mientras una condición dada fuera cierta. Pero, ¿qué ocurre con programas más complicados en los que muchos sucesos diferentes pueden hacer que el programa deje de ejecutarse?

In the previous example, we had the program perform certain tasks while a given condition was true. But what about more complicated programs in which many different events could cause the program to stop running?

Por ejemplo, en un juego, varios sucesos diferentes pueden poner fin a la partida. Cuando el jugador se queda sin naves, se le acaba el tiempo o las ciudades que debía proteger son destruidas, el juego debe terminar. Debe terminar si ocurre cualquiera de estos acontecimientos. Si pueden ocurrir muchos eventos posibles para detener el programa, intentar probar todas estas condiciones en una sentencia `while` se vuelve complicado y difícil.

For example, in a game, several different events can end the game. When the player runs out of ships, their time runs out, or the cities they were supposed to protect are all destroyed, the game should end. It needs to end if any one of these events happens. If many possible events might occur to stop the program, trying to test all these conditions in one `while` statement becomes complicated and difficult.

Para un programa que debe ejecutarse sólo mientras muchas condiciones sean verdaderas, puedes definir una variable que determine si todo el programa está activo o no. Esta variable, llamada *flag*, actúa como una señal para el programa. Podemos escribir nuestros programas de modo que se ejecuten mientras la bandera esté establecida en `True` y dejen de ejecutarse cuando alguno de varios eventos establezca el valor de la bandera en `False`. Como resultado, nuestra declaración global `while` sólo necesita comprobar una condición: si la bandera está actualmente en `True`. Entonces, todas nuestras demás pruebas (para ver si se ha producido un evento que debería poner la bandera en `False`) pueden organizarse ordenadamente en el resto del programa.

For a program that should run only as long as many conditions are true, you can define one variable that determines whether or not the entire program is active. This variable, called a *flag*, acts as a signal to the program. We can write our programs so they run while the flag is set to `True` and stop running when any of several events sets the value of the flag to `False`. As a result, our overall `while` statement needs to check only one condition: whether the flag is currently `True`. Then, all our other tests (to see if an event has occurred that should set the flag to `False`) can be neatly organized in the rest of the program.

Añadamos una bandera a *parrot.py* de la sección anterior. Esta bandera, que llamaremos `active` (aunque puedes llamarla como quieras), controlará si el programa debe seguir ejecutándose o no:

Let's add a flag to *parrot.py* from the previous section. This flag, which we'll call `active` (though you can call it anything), will monitor whether or not the program should continue running:

```
prompt = "\nTell me something, and I will repeat it back to you:"
prompt += "\nEnter 'quit' to end the program. "
```

```
active = True
❶ while active:
    message = input(prompt)

    if message == 'quit':
        active = False
    else:
        print(message)
```

Establecemos la variable `active` en `True` para que el programa comience en un estado activo. Hacer esto simplifica la sentencia `while` porque no se realiza ninguna comparación en la propia sentencia `while`; de la lógica se encargan otras partes del programa. Mientras la variable `active` siga siendo `True`, el bucle continuará ejecutándose ❶.

We set the variable `active` to `True` so the program starts in an active state. Doing so makes the `while` statement simpler because no comparison is made in the `while`

statement itself; the logic is taken care of in other parts of the program. As long as the active variable remains `True` , the loop will continue running ❶ .

En la sentencia `if` dentro del bucle `while` , comprobamos el valor de `message` una vez que el usuario introduce su entrada. Si el usuario introduce `'quit'` , establecemos `active` en `False` , y el bucle `while` se detiene. Si el usuario introduce algo distinto de `'quit'` , imprimimos su entrada como mensaje.

In the `if` statement inside the `while` loop, we check the value of `message` once the user enters their input. If the user enters `'quit'` , we set `active` to `False` , and the `while` loop stops. If the user enters anything other than `'quit'` , we print their input as a message.

Este programa tiene la misma salida que el ejemplo anterior, en el que colocamos la prueba condicional directamente en la sentencia `while` . Pero ahora que tiene una bandera para indicar si el programa en general está en estado activo, sería fácil añadir más pruebas (como las sentencias `elif`) para los eventos que deberían hacer que `active` se convirtiera en `False` . Esto es útil en programas complicados como los juegos, en los que puede haber muchos eventos que deberían hacer que el programa dejara de ejecutarse. Cuando alguno de estos eventos haga que la bandera activa se convierta en `False` , el bucle principal del juego saldrá, se podrá mostrar un mensaje *Game Over* y se podrá dar al jugador la opción de volver a jugar.

This program has the same output as the previous example where we placed the conditional test directly in the `while` statement. But now that we have a flag to indicate whether the overall program is in an active state, it would be easy to add more tests (such as `elif` statements) for events that should cause `active` to become `False` . This is useful in complicated programs like games, in which there may be many events that should each make the program stop running. When any of these events causes the active flag to become `False` , the main game loop will exit, a *Game Over* message can be displayed, and the player can be given the option to play again.

Usar `break` para salir de un bucle (Using `break` to Exit a Loop)

Para salir inmediatamente de un bucle `while` sin ejecutar ningún código restante en el bucle, independientemente de los resultados de cualquier prueba condicional, utiliza la sentencia `break` . La sentencia `break` dirige el flujo de tu programa; puedes utilizarla para controlar qué líneas de código se ejecutan y cuáles no, de modo que el programa sólo

ejecute el código que tú quieras, cuando tú quieras.

To exit a `while` loop immediately without running any remaining code in the loop, regardless of the results of any conditional test, use the `break` statement. The `break` statement directs the flow of your program; you can use it to control which lines of code are executed and which aren't, so the program only executes code that you want it to, when you want it to.

Por ejemplo, considera un programa que pregunta al usuario sobre los lugares que ha visitado. Podemos detener el bucle `while` en este programa llamando a `break` en cuanto el usuario introduzca el valor `'quit'` :

For example, consider a program that asks the user about places they've visited. We can stop the `while` loop in this program by calling `break` as soon as the user enters the `'quit'` value:

ciudades.py

cities.py

```
prompt = "\nPlease enter the name of a city you have visited:"  
prompt += "\n(Enter 'quit' when you are finished.) "
```

```
❶ while True:  
    city = input(prompt)  
  
    if city == 'quit':  
        break  
    else:  
        print(f'I'd love to go to {city.title()}!')
```

Un bucle que comienza con `while True` ❶ se ejecutará eternamente a menos que llegue a una sentencia `break` . El bucle de este programa sigue pidiendo al usuario que introduzca los nombres de las ciudades en las que ha estado hasta que introduzca `'quit'` . Cuando introduzcan `'quit'` , se ejecutará la sentencia `break` , haciendo que Python salga del bucle:

A loop that starts with `while True` ❶ will run forever unless it reaches a `break` statement. The loop in this program continues asking the user to enter the names of cities they've been to until they enter `'quit'`. When they enter `'quit'`, the `break` statement runs, causing Python to exit the loop:

Please enter the name of a city you have visited:
(Enter 'quit' when you are finished.) New York
I'd love to go to New York!

Please enter the name of a city you have visited:
(Enter 'quit' when you are finished.) San Francisco
I'd love to go to San Francisco!

Please enter the name of a city you have visited:
(Enter 'quit' when you are finished.) quit

Nota (Note)

Puedes utilizar la sentencia `break` en cualquiera de los bucles de Python. Por ejemplo, podrías utilizar `break` para salir de un bucle `for` que esté trabajando a través de una lista o un diccionario.

You can use the `break` statement in any of Python's loops. For example, you could use `break` to quit a `for` loop that's working through a list or a dictionary.

Utilizar `continue` en un bucle (Using `continue` in a Loop)

En lugar de salir completamente de un bucle sin ejecutar el resto de su código, puedes utilizar la sentencia `continue` para volver al principio del bucle, basándote en el resultado de una prueba condicional. Por ejemplo, considera un bucle que cuenta del 1 al 10, pero que sólo imprime los números impares de ese rango:

Rather than breaking out of a loop entirely without executing the rest of its code, you can use the `continue` statement to return to the beginning of the loop, based on the result of a conditional test. For example, consider a loop that counts from 1 to 10 but prints only the odd numbers in that range:

contando.py

counting.py

```
current_number = 0
while current_number < 10:
    ❶ current_number += 1
    if current_number % 2 == 0:
        continue

    print(current_number)
```

Primero, ponemos `current_number` a 0. Como es menor que 10, Python entra en el bucle `while`. Una vez dentro del bucle, incrementamos la cuenta en 1 ❶, por lo que `current_number` es 1. A continuación, la sentencia `if` comprueba el módulo de `current_number` y 2. Si el módulo es 0 (lo que significa que `current_number` es divisible por 2), la sentencia `continue` indica a Python que ignore el resto del bucle y vuelva al principio. Si el número actual no es divisible por 2, se ejecuta el resto del bucle y Python imprime el número actual:

First, we set `current_number` to 0. Because it's less than 10, Python enters the `while` loop. Once inside the loop, we increment the count by 1 ❶, so `current_number` is 1. The `if` statement then checks the modulo of `current_number` and 2. If the modulo is 0 (which means `current_number` is divisible by 2), the `continue` statement tells Python to ignore the rest of the loop and return to the beginning. If the current number is not divisible by 2, the rest of the loop is executed and Python prints the current number:

```
1
3
5
7
9
```

Evitar bucles infinitos (*Avoiding Infinite Loops*)

Todo bucle `while` necesita una forma de dejar de ejecutarse para que no siga ejecutándose eternamente. Por ejemplo, este bucle de conteo debe contar de 1 a 5:

Every `while` loop needs a way to stop running so it won't continue to run forever. For example, this counting loop should count from 1 to 5:

contando.py

counting.py

```
x = 1
while x <= 5:
    print(x)
    x += 1
```

Sin embargo, si accidentalmente omites la línea `x += 1` , el bucle se ejecutará eternamente:

However, if you accidentally omit the line `x += 1` , the loop will run forever:

```
# This loop runs forever!
x = 1
while x <= 5:
    print(x)
```

Ahora el valor de `x` comenzará en 1 pero nunca cambiará. Como resultado, la prueba condicional `x <= 5` siempre se evaluará a `True` y el bucle `while` se ejecutará eternamente, imprimiendo una serie de 1s, como esto:

Now the value of `x` will start at 1 but never change. As a result, the conditional test `x <= 5` will always evaluate to `True` and the `while` loop will run forever, printing a series of 1s, like this:

```
1
1
1
1
--snip--
```

Todo programador escribe accidentalmente un bucle infinito `while` de vez en cuando, especialmente cuando los bucles de un programa tienen sutiles condiciones de salida. Si tu programa se queda atascado en un bucle infinito, pulsa CTRL-C o simplemente cierra la ventana del terminal que muestra la salida de tu programa.

Every programmer accidentally writes an infinite `while` loop from time to time, especially when a program's loops have subtle exit conditions. If your program gets stuck in an infinite loop, press CTRL-C or just close the terminal window displaying your program's output.

Para evitar escribir bucles infinitos, prueba cada bucle de `while` y asegúrate de que el bucle se detiene cuando esperas que lo haga. Si quieres que tu programa termine cuando el usuario introduzca un determinado valor de entrada, ejecuta el programa e introduce ese valor. Si el programa no finaliza, examina la forma en que tu programa maneja el valor que debería provocar la salida del bucle. Asegúrate de que al menos una parte del programa puede hacer que la condición del bucle `False` o hacer que llegue a una sentencia `break` .

To avoid writing infinite loops, test every `while` loop and make sure the loop stops when you expect it to. If you want your program to end when the user enters a certain input value, run the program and enter that value. If the program doesn't end, scrutinize the way your program handles the value that should cause the loop to exit. Make sure at least one part of the program can make the loop's condition `False` or cause it to reach a `break` statement.

Nota (Note)

VS Code, como muchos editores, muestra la salida en una ventana de terminal incrustada. Para cancelar un bucle infinito, asegúrate de hacer clic en el área de salida del editor antes de pulsar CTRL-C.

VS Code, like many editors, displays output in an embedded terminal window. To cancel an infinite loop, make sure you click in the output area of the editor before pressing CTRL-C.

Pruébalo tú mismo *(Try It Yourself)*

7-4. Pizza Toppings: Escribe un bucle que pida al usuario que introduzca una serie de ingredientes de pizza hasta que introduzca un valor 'quit' . A medida que introduzcan cada ingrediente, imprime un mensaje diciendo que añadirás ese ingrediente a su pizza.

7-4. Pizza Toppings: Write a loop that prompts the user to enter a series of pizza toppings until they enter a 'quit' value. As they enter each topping, print a message saying you'll add that topping to their pizza.

7-5. Entradas de cine: Un cine cobra diferentes precios de entrada según la edad de la persona. Si una persona tiene menos de 3 años, la entrada es gratis; si tiene entre 3 y 12 años, la entrada cuesta 10\$; y si tiene más de 12 años, la entrada cuesta 15\$. Escribe un bucle en el que preguntes a los usuarios su edad, y luego les digas el precio de su entrada de cine.

7-5. Movie Tickets: A movie theater charges different ticket prices depending on a person's age. If a person is under the age of 3, the ticket is free; if they are between 3 and 12, the ticket is \$10; and if they are over age 12, the ticket is \$15. Write a loop in which you ask users their age, and then tell them the cost of their movie ticket.

7-6. Tres salidas: Escribe diferentes versiones del Ejercicio 7-4 o 7-5 que hagan cada una de las siguientes cosas al menos una vez:

7-6. Three Exits: Write different versions of either Exercise 7-4 or 7-5 that do each of the following at least once:

Listado en español:

- Utilizar una prueba condicional en la sentencia `while` para detener el bucle.
- Utilizar una variable `active` para controlar la duración del bucle.
- Utiliza una sentencia `break` para salir del bucle cuando el usuario introduzca un valor `'quit'` .

Listing in English:

- Use a conditional test in the `while` statement to stop the loop.
- Use an `active` variable to control how long the loop runs.
- Use a `break` statement to exit the loop when the user enters a `'quit'` value.

7-7. Infinito: Escribe un bucle que no termine nunca y ejecútalo. (Para terminar el bucle, pulsa CTRL-C o cierra la ventana que muestra la salida).

7-7. Infinity: Write a loop that never ends, and run it. (To end the loop, press CTRL-C or close the window displaying the output.)

Utilizar un bucle `while` con listas y diccionarios (*Using a while Loop with Lists and Dictionaries*)

Hasta ahora, sólo hemos trabajado con una información del usuario cada vez. Recibimos la entrada del usuario y luego imprimimos la entrada o una respuesta a ella. La siguiente vez que pasáramos por el bucle `while` , recibiríamos otro valor de entrada y responderíamos a él. Pero para hacer un seguimiento de muchos usuarios y piezas de información, necesitaremos utilizar listas y diccionarios con nuestros bucles `while` .

So far, we've worked with only one piece of user information at a time. We received the user's input and then printed the input or a response to it. The next time through the `while` loop, we'd receive another input value and respond to that. But to keep track of many users and pieces of information, we'll need to use lists and dictionaries with our `while` loops.

Un bucle `for` es eficaz para recorrer una lista, pero no debes modificar una lista dentro de un bucle `for` porque Python tendrá problemas para seguir la pista de los elementos de la lista. Para modificar una lista a medida que la recorres, utiliza un bucle `while` . Utilizar bucles `while` con listas y diccionarios te permite recopilar, almacenar y organizar montones de datos de entrada para examinarlos e informar sobre ellos más tarde.

A `for` loop is effective for looping through a list, but you shouldn't modify a list inside a `for` loop because Python will have trouble keeping track of the items in the list. To modify a list as you work through it, use a `while` loop. Using `while` loops with lists and dictionaries allows you to collect, store, and organize lots of input to examine and report on later.

Mover elementos de una lista a otra (*Moving Items from One List to Another*)

Considera una lista de usuarios recién registrados pero no verificados de un sitio web. Después de verificar a estos usuarios, ¿cómo podemos moverlos a otra lista de usuarios confirmados? Una forma sería utilizar un bucle `while` para extraer usuarios de la lista de usuarios no confirmados a medida que los verificamos y luego añadirlos a una lista separada de usuarios confirmados. Éste es el aspecto que podría tener ese código:

Consider a list of newly registered but unverified users of a website. After we verify these users, how can we move them to a separate list of confirmed users? One way would be to use a `while` loop to pull users from the list of unconfirmed users as we verify them and then add them to a separate list of confirmed users. Here's what that code might look like:

usuarios_confirmados.py

confirmed_users.py

```
# Start with users that need to be verified,
# and an empty list to hold confirmed users.
❶ unconfirmed_users = ['alice', 'brian', 'candace']
confirmed_users = []

# Verify each user until there are no more unconfirmed users.
# Move each verified user into the list of confirmed users.
❷ while unconfirmed_users:
    ❸ current_user = unconfirmed_users.pop()

    print(f"Verifying user: {current_user.title()}")
    ❹ confirmed_users.append(current_user)
```



```
# Display all confirmed users.  
print("\nThe following users have been confirmed:")  
for confirmed_user in confirmed_users:  
    print(confirmed_user.title())
```

Empezamos con una lista de usuarios no confirmados ❶ (Alice, Brian y Candace) y una lista vacía para contener los usuarios confirmados. El bucle `while` se ejecuta mientras la lista `unconfirmed_users` no esté vacía ❷. Dentro de este bucle, el método `pop()` elimina a los usuarios no confirmados de uno en uno desde el final de `unconfirmed_users` ❸. Como Candace es la última de la lista `unconfirmed_users`, su nombre será el primero en ser eliminado, asignado a `current_user`, y añadido a la lista `confirmed_users` ❹. El siguiente es Brian, y después Alice.

We begin with a list of unconfirmed users ❶ (Alice, Brian, and Candace) and an empty list to hold confirmed users. The `while` loop runs as long as the list `unconfirmed_users` is not empty ❷. Within this loop, the `pop()` method removes unverified users one at a time from the end of `unconfirmed_users` ❸. Because Candace is last in the `unconfirmed_users` list, her name will be the first to be removed, assigned to `current_user`, and added to the `confirmed_users` list ❹. Next is Brian, then Alice.

Simulamos la confirmación de cada usuario imprimiendo un mensaje de verificación y añadiéndolo después a la lista de usuarios confirmados. A medida que se reduce la lista de usuarios no confirmados, crece la lista de usuarios confirmados. Cuando la lista de usuarios no confirmados esté vacía, el bucle se detiene y se imprime la lista de usuarios confirmados:

We simulate confirming each user by printing a verification message and then adding them to the list of confirmed users. As the list of unconfirmed users shrinks, the list of confirmed users grows. When the list of `unconfirmed_users` is empty, the loop stops and the list of confirmed users is printed:

```
Verifying user: Candace  
Verifying user: Brian  
Verifying user: Alice
```

```
The following users have been confirmed:  
Candace  
Brian  
Alice
```

Eliminar todas las instancias de valores específicos de una lista *(Removing All Instances of Specific Values from a List)*

En el Capítulo 3, utilizamos `remove()` para eliminar un valor específico de una lista. La función `remove()` funcionó porque el valor que nos interesaba sólo aparecía una vez en la lista. Pero, ¿qué ocurre si quieres eliminar todas las instancias de un valor de una lista?

In Chapter 3, we used `remove()` to remove a specific value from a list. The `remove()` function worked because the value we were interested in appeared only once in the list. But what if you want to remove all instances of a value from a list?

Supongamos que tienes una lista de mascotas con el valor `'cat'` repetido varias veces. Para eliminar todas las instancias de ese valor, puedes ejecutar un bucle `while` hasta que `'cat'` deje de estar en la lista, como se muestra aquí:

Say you have a list of pets with the value `'cat'` repeated several times. To remove all instances of that value, you can run a `while` loop until `'cat'` is no longer in the list, as shown here:

mascotas.py

pets.py

```
pets = ['dog', 'cat', 'dog', 'goldfish', 'cat', 'rabbit', 'cat']  
print(pets)
```

```
while 'cat' in pets:  
    pets.remove('cat')
```

```
print(pets)
```

Empezamos con una lista que contiene varias instancias de `'cat'`. Tras imprimir la lista, Python entra en el bucle `while` porque encuentra el valor `'cat'` en la lista al menos una vez. Una vez dentro del bucle, Python elimina la primera instancia de `'cat'`, vuelve a la línea `while` y vuelve a entrar en el bucle cuando encuentra que `'cat'` sigue en la lista. Elimina cada instancia de `'cat'` hasta que el valor deje de estar en la lista, momento en el

que Python sale del bucle y vuelve a imprimir la lista:

We start with a list containing multiple instances of 'cat' . After printing the list, Python enters the `while` loop because it finds the value 'cat' in the list at least once. Once inside the loop, Python removes the first instance of 'cat' , returns to the `while` line, and then reenters the loop when it finds that 'cat' is still in the list. It removes each instance of 'cat' until the value is no longer in the list, at which point Python exits the loop and prints the list again:

```
['dog', 'cat', 'dog', 'goldfish', 'cat', 'rabbit', 'cat']  
['dog', 'dog', 'goldfish', 'rabbit']
```

Rellenar un diccionario con datos introducidos por el usuario (*Filling a Dictionary with User Input*)

Puedes pedir tantas entradas como necesites en cada pasada por un bucle `while` . Hagamos un programa de encuestas en el que en cada paso por el bucle se pida el nombre y la respuesta del participante. Almacenaremos los datos que recojamos en un diccionario, porque queremos relacionar cada respuesta con un usuario concreto:

You can prompt for as much input as you need in each pass through a `while` loop. Let's make a polling program in which each pass through the loop prompts for the participant's name and response. We'll store the data we gather in a dictionary, because we want to connect each response with a particular user:

encuesta_montaña.py

mountain_poll.py

```
responses = {}  
# Set a flag to indicate that polling is active.  
polling_active = True  
  
while polling_active:  
    # Prompt for the person's name and response.  
    ❶ name = input("\nWhat is your name? ")
```

```

response = input("Which mountain would you like to climb someday? ")

# Store the response in the dictionary.
❷ responses[name] = response

# Find out if anyone else is going to take the poll.
❸ repeat = input("Would you like to let another person respond? (yes/ no) ")
if repeat == 'no':
    polling_active = False

# Polling is complete. Show the results.
print("\n--- Poll Results ---")
❹ for name, response in responses.items():
    print(f'{name} would like to climb {response}.')

```

El programa define primero un diccionario vacío (`responses`) y establece una bandera (`polling_active`) para indicar que el sondeo está activo. Mientras `polling_active` sea `True` , Python ejecutará el código en el bucle `while` .

The program first defines an empty dictionary (`responses`) and sets a flag (`polling_active`) to indicate that polling is active. As long as `polling_active` is `True` , Python will run the code in the `while` loop.

Dentro del bucle, se pide al usuario que introduzca su nombre y la montaña que le gustaría escalar ❶ . Esa información se almacena en el diccionario `responses` ❷ , y se pregunta al usuario si desea o no seguir con el sondeo ❸ . Si entra `yes` , el programa vuelve a entrar en el bucle `while` . Si entra `no` , la bandera `polling_active` se pone en `False` , el bucle `while` deja de ejecutarse, y el bloque de código final ❹ muestra los resultados del sondeo.

Within the loop, the user is prompted to enter their name and a mountain they'd like to climb ❶ . That information is stored in the `responses` dictionary ❷ , and the user is asked whether or not to keep the poll running ❸ . If they enter `yes` , the program enters the `while` loop again. If they enter `no` , the `polling_active` flag is set to `False` , the `while` loop stops running, and the final code block ❹ displays the results of the poll.

Si ejecutas este programa e introduces respuestas de muestra, deberías ver una salida como ésta:

If you run this program and enter sample responses, you should see output like this:

What is your name? Eric
Which mountain would you like to climb someday? Denali
Would you like to let another person respond? (yes/ no) yes

What is your name? Lynn
Which mountain would you like to climb someday? Devil's Thumb
Would you like to let another person respond? (yes/ no) no

--- Poll Results ---
Eric would like to climb Denali.
Lynn would like to climb Devil's Thumb.

Pruébalo tú mismo (Try It Yourself)

7-8. Deli: Haz una lista llamada `sandwich_orders` y rellénala con los nombres de varios bocadillos. Luego haz una lista vacía llamada `finished_sandwiches`. Recorre la lista de pedidos de bocadillos e imprime un mensaje para cada pedido, como `I made your tuna sandwich`. A medida que se hace cada bocadillo, muévelo a la lista de bocadillos terminados. Cuando se hayan hecho todos los bocadillos, imprime un mensaje con la lista de cada bocadillo hecho.

7-8. Deli: Make a list called `sandwich_orders` and fill it with the names of various sandwiches. Then make an empty list called `finished_sandwiches`. Loop through the list of sandwich orders and print a message for each order, such as `I made your tuna sandwich`. As each sandwich is made, move it to the list of finished sandwiches. After all the sandwiches have been made, print a message listing each sandwich that was made.

7-9. No hay Pastrami: Utilizando la lista `sandwich_orders` del Ejercicio 7-8, asegúrate de que el bocadillo 'pastrami' aparece en la lista al menos tres veces. Añade código cerca del principio de tu programa para imprimir un mensaje diciendo que la charcutería se ha quedado sin pastrami, y luego utiliza un bucle `while` para eliminar todas las apariciones de 'pastrami' de `sandwich_orders`. Asegúrate de que ningún bocadillo de pastrami acaba en `finished_sandwiches`.

7-9. No Pastrami: Using the list `sandwich_orders` from Exercise 7-8, make sure the sandwich 'pastrami' appears in the list at least three times. Add code near the beginning of your program to print a message saying the deli has run out of pastrami, and then use a `while` loop to remove all occurrences of 'pastrami' from `sandwich_orders`. Make sure no pastrami sandwiches end up in `finished_sandwiches`.

7-10. Vacaciones de ensueño: Escribe un programa que sondee a los usuarios sobre sus vacaciones soñadas. Escribe una pregunta similar a *If you could visit one place in the world, where would you go?* Incluye un bloque de código que imprima los resultados de la encuesta.

7-10. Dream Vacation: Write a program that polls users about their dream vacation. Write a prompt similar to *If you could visit one place in the world, where would you go?* Include a block of code that prints the results of the poll.

Resumen (Summary)

En este capítulo, has aprendido a utilizar `input()` para permitir que los usuarios proporcionen su propia información en tus programas. Aprendiste a trabajar tanto con entradas de texto como numéricas y a utilizar los bucles `while` para que tus programas se ejecuten tanto tiempo como quieran tus usuarios. Viste varias formas de controlar el flujo de un bucle `while` estableciendo una bandera `active`, utilizando la sentencia `break` y utilizando la sentencia `continue`. Aprendiste a utilizar un bucle `while` para mover elementos de una lista a otra y a eliminar todas las instancias de un valor de una lista. También aprendiste cómo se pueden utilizar los bucles `while` con los diccionarios.

In this chapter, you learned how to use `input()` to allow users to provide their own information in your programs. You learned to work with both text and numerical input and how to use `while` loops to make your programs run as long as your users want them to. You saw several ways to control the flow of a `while` loop by setting an `active` flag, using the `break` statement, and using the `continue` statement. You learned how to use a `while` loop to move items from one list to another and how to remove all instances of a value from a list. You also learned how `while` loops can be used with dictionaries.

En el Capítulo 8 aprenderás sobre las funciones. *Functions* te permiten dividir tus programas en pequeñas partes, cada una de las cuales realiza un trabajo específico. Puedes llamar a una función tantas veces como quieras, y puedes almacenar tus funciones en archivos separados. Utilizando funciones, podrás escribir un código más eficiente, más fácil de solucionar y mantener, y que puede reutilizarse en muchos programas diferentes.

In Chapter 8 you'll learn about functions. *Functions* allow you to break your programs into small parts, each of which does one specific job. You can call a function as many times as you want, and you can store your functions in separate files. By using functions, you'll be able to write more efficient code that's easier to troubleshoot and maintain and that can be reused in many different programs.

Capítulo 8: Funciones (Chapter 8: Functions)

8

Funciones (8 Functions)



DESCRIPCIÓN DE LA
IMAGEN (En español):

La imagen muestra una ilustración de una serpiente enrollada alrededor de un motor. La serpiente tiene una expresión determinada en su rostro, con cejas furiosas y unos ojos intensos. El diseño es en blanco y negro, con un enfoque en los detalles de las escamas de la serpiente y las aspas del motor. La escena se encuentra dentro de un círculo, resaltando la unión de la naturaleza con la tecnología.

En este capítulo aprenderás a escribir *functions* , que son bloques de código con nombre diseñados para realizar una tarea específica. Cuando quieras realizar una tarea concreta que hayas definido en una función, escribe en *call* la función responsable de ella. Si necesitas realizar esa tarea varias veces a lo largo de tu programa, no necesitas escribir todo el código para la misma tarea una y otra vez; sólo tienes que llamar a la función dedicada a gestionar esa tarea, y la llamada indica a Python que ejecute el código dentro de la función. Verás que el uso de funciones hace que tus programas sean más fáciles de escribir, leer, probar y corregir.

In this chapter you'll learn to write *functions* , which are named blocks of code designed to do one specific job. When you want to perform a particular task that you've defined in a function, you *call* the function responsible for it. If you need to perform that task multiple times throughout your program, you don't need to type all the code for the same task again and again; you just call the function dedicated to handling that task, and the call tells

Python to run the code inside the function. You'll find that using functions makes your programs easier to write, read, test, and fix.

En este capítulo también aprenderás diversas formas de pasar información a las funciones. Aprenderás a escribir ciertas funciones cuyo trabajo principal es mostrar información y otras funciones diseñadas para procesar datos y devolver un valor o conjunto de valores. Por último, aprenderás a almacenar las funciones en archivos independientes llamados *modules* para ayudarte a organizar los archivos principales de tu programa.

In this chapter you'll also learn a variety of ways to pass information to functions. You'll learn how to write certain functions whose primary job is to display information and other functions designed to process data and return a value or set of values. Finally, you'll learn to store functions in separate files called *modules* to help organize your main program files.

Definir una función (*Defining a Function*)

Aquí tienes una función sencilla llamada `greet_user()` que imprime un saludo:

Here's a simple function named `greet_user()` that prints a greeting:

saludador.py

greeter.py

```
def greet_user():
    """Display a simple greeting."""
    print("Hello!")
```

```
greet_user()
```

Este ejemplo muestra la estructura más simple de una función. La primera línea utiliza la palabra clave `def` para informar a Python de que estás definiendo una función. Se trata de *function definition*, que indica a Python el nombre de la función y, si procede, qué tipo de información necesita la función para hacer su trabajo. Los paréntesis contienen esa información. En este caso, el nombre de la función es `greet_user()`, y no necesita

información para realizar su trabajo, por lo que sus paréntesis están vacíos. (Aún así, los paréntesis son necesarios.) Por último, la definición termina con dos puntos.

This example shows the simplest structure of a function. The first line uses the keyword `def` to inform Python that you're defining a function. This is the *function definition*, which tells Python the name of the function and, if applicable, what kind of information the function needs to do its job. The parentheses hold that information. In this case, the name of the function is `greet_user()`, and it needs no information to do its job, so its parentheses are empty. (Even so, the parentheses are required.) Finally, the definition ends in a colon.

Las líneas sangradas que siguen a `def greet_user()`: constituyen la *body* de la función. El texto de la segunda línea es un comentario llamado *docstring*, que describe lo que hace la función. Cuando Python genera documentación para las funciones de tus programas, busca una cadena inmediatamente después de la definición de la función. Estas cadenas suelen ir entre comillas triples, lo que te permite escribir varias líneas.

Any indented lines that follow `def greet_user()`: make up the *body* of the function. The text on the second line is a comment called a *docstring*, which describes what the function does. When Python generates documentation for the functions in your programs, it looks for a string immediately after the function's definition. These strings are usually enclosed in triple quotes, which lets you write multiple lines.

La línea `print("Hello!")` es la única línea de código real en el cuerpo de esta función, por lo que `greet_user()` sólo tiene un trabajo: `print("Hello!")`.

The line `print("Hello!")` is the only line of actual code in the body of this function, so `greet_user()` has just one job: `print("Hello!")`.

Cuando quieras utilizar esta función, tienes que llamarla. Un *function call* indica a Python que ejecute el código de la función. Para *call* una función, escribe el nombre de la función, seguido de cualquier información necesaria entre paréntesis. Como aquí no se necesita ninguna información, llamar a nuestra función es tan sencillo como escribir `greet_user()`. Como era de esperar, imprime `Hello!`:

When you want to use this function, you have to call it. A *function call* tells Python to execute the code in the function. To *call* a function, you write the name of the function, followed by any necessary information in parentheses. Because no information is needed here, calling our function is as simple as entering `greet_user()` . As expected, it prints Hello! :

Hello!

Pasar información a una función (*Passing Information to a Function*)

Si modificas ligeramente la función `greet_user()` , puede saludar al usuario por su nombre. Para que la función lo haga, introduce `username` entre los paréntesis de la definición de la función en `def greet_user()` . Añadiendo aquí `username` , permites que la función acepte cualquier valor de `username` que especifiques. Ahora la función espera que proporciones un valor para `username` cada vez que la llames. Cuando llames a `greet_user()` , puedes pasarle un nombre, como `'jesse'` , dentro de los paréntesis:

If you modify the function `greet_user()` slightly, it can greet the user by name. For the function to do this, you enter `username` in the parentheses of the function's definition at `def greet_user()` . By adding `username` here, you allow the function to accept any value of `username` you specify. The function now expects you to provide a value for `username` each time you call it. When you call `greet_user()` , you can pass it a name, such as `'jesse'` , inside the parentheses:

```
def greet_user(username):  
    """Display a simple greeting."""  
    print(f"Hello, {username.title()}!")
```

```
greet_user('jesse')
```

Al introducir `greet_user('jesse')` se llama a `greet_user()` y se proporciona a la función la información que necesita para ejecutar la llamada a `print()` . La función acepta el nombre que le has pasado y muestra el saludo correspondiente a ese nombre:

Entering `greet_user('jesse')` calls `greet_user()` and gives the function the information it needs to execute the `print()` call. The function accepts the name you passed it and displays the greeting for that name:

Hello, Jesse!

Del mismo modo, al introducir `greet_user('sarah')` se llama a `greet_user()` , se le pasa 'sarah' , y se imprime Hello, Sarah! Puedes llamar a `greet_user()` tantas veces como quieras y pasarle cualquier nombre que desees para producir una salida predecible cada vez.

Likewise, entering `greet_user('sarah')` calls `greet_user()` , passes it 'sarah' , and prints Hello, Sarah! You can call `greet_user()` as often as you want and pass it any name you want to produce a predictable output every time.

Argumentos y parámetros (*Arguments and Parameters*)

En la función `greet_user()` anterior, definimos `greet_user()` para requerir un valor para la variable `username` . Una vez que llamamos a la función y le dimos la información (el nombre de una persona), imprimió el saludo correcto.

In the preceding `greet_user()` function, we defined `greet_user()` to require a value for the variable `username` . Once we called the function and gave it the information (a person's name), it printed the right greeting.

La variable `username` en la definición de `greet_user()` es un ejemplo de *parameter* , un dato que la función necesita para hacer su trabajo. El valor 'jesse' en `greet_user('jesse')` es un ejemplo de argumento. Un *argument* es un fragmento de información que se pasa de una llamada a una función. Cuando llamamos a la función, colocamos entre paréntesis el valor con el que queremos que trabaje la función. En este caso, el argumento 'jesse' se pasó a la función `greet_user()` , y el valor se asignó al parámetro `username` .

The variable `username` in the definition of `greet_user()` is an example of a *parameter* , a piece of information the function needs to do its job. The value 'jesse' in `greet_user('jesse')` is an example of an argument. An *argument* is a piece of information that's passed from a function call to a function. When we call the function, we place the value we want the function to work with in parentheses. In this case the argument 'jesse' was passed to the function `greet_user()` , and the value was assigned to the parameter `username` .

Nota (Note)

A veces se habla indistintamente de argumentos y parámetros. No te sorprendas si ves que las variables de una definición de función se denominan argumentos o que las variables de una llamada a una función se denominan parámetros.

People sometimes speak of arguments and parameters interchangeably. Don't be surprised if you see the variables in a function definition referred to as arguments or the variables in a function call referred to as parameters.

Pruébalo tú mismo (Try It Yourself)

8-1. Mensaje: Escribe una función llamada `display_message()` que imprima una frase contando a todo el mundo lo que estás aprendiendo en este capítulo. Llama a la función y asegúrate de que el mensaje se muestra correctamente.

8-1. Message: Write a function called `display_message()` that prints one sentence telling everyone what you are learning about in this chapter. Call the function, and make sure the message displays correctly.

8-2. Libro favorito: Escribe una función llamada `favorite_book()` que acepte un parámetro, `title`. La función debe imprimir un mensaje, como `One of my favorite books is Alice in Wonderland`. Llama a la función, asegurándote de incluir el título de un libro como argumento en la llamada a la función.

8-2. Favorite Book: Write a function called `favorite_book()` that accepts one parameter, `title`. The function should print a message, such as `One of my favorite books is Alice in Wonderland`. Call the function, making sure to include a book title as an argument in the function call.

Pasar argumentos (*Passing Arguments*)

Dado que la definición de una función puede tener varios parámetros, la llamada a una función puede necesitar varios argumentos. Puedes pasar argumentos a tus funciones de varias formas. Puedes utilizar *positional arguments*, que deben estar en el mismo orden en que se escribieron los parámetros; *keyword arguments*, donde cada argumento consta de un nombre de variable y un valor; y listas y diccionarios de valores. Veamos cada uno de ellos por separado.

Because a function definition can have multiple parameters, a function call may need multiple arguments. You can pass arguments to your functions in a number of ways. You can use *positional arguments*, which need to be in the same order the parameters were written; *keyword arguments*, where each argument consists of a variable name and a value; and lists and dictionaries of values. Let's look at each of these in turn.

Argumentos posicionales (*Positional Arguments*)

Cuando llamas a una función, Python debe hacer coincidir cada argumento de la llamada a la función con un parámetro de la definición de la función. La forma más sencilla de hacerlo es basándose en el orden de los argumentos proporcionados. Los valores emparejados de esta forma se denominan *positional arguments*.

When you call a function, Python must match each argument in the function call with a parameter in the function definition. The simplest way to do this is based on the order of the arguments provided. Values matched up this way are called *positional arguments*.

Para ver cómo funciona esto, considera una función que muestra información sobre mascotas. La función nos dice qué tipo de animal es cada mascota y el nombre de la mascota, como se muestra aquí:

To see how this works, consider a function that displays information about pets. The function tells us what kind of animal each pet is and the pet's name, as shown here:

mascotas.py

pets.py

```
❶ def describe_pet(animal_type, pet_name):  
    """Display information about a pet."""  
    print(f"\nI have a {animal_type}.")  
    print(f"My {animal_type}'s name is {pet_name.title().}")  
  
❷ describe_pet('hamster', 'harry')
```

La definición muestra que esta función necesita un tipo de animal y el nombre del animal ❶ . Cuando llamamos a `describe_pet()` , necesitamos proporcionar un tipo de animal y un nombre, en ese orden. Por ejemplo, en la llamada a la función, el argumento `'hamster'` se asigna al parámetro `animal_type` y el argumento `'harry'` se asigna al parámetro `pet_name` ❷ . En el cuerpo de la función, estos dos parámetros se utilizan para mostrar información sobre la mascota que se está describiendo.

The definition shows that this function needs a type of animal and the animal's name ❶ . When we call `describe_pet()` , we need to provide an animal type and a name, in that order. For example, in the function call, the argument `'hamster'` is assigned to the parameter `animal_type` and the argument `'harry'` is assigned to the parameter `pet_name` ❷ . In the function body, these two parameters are used to display information about the pet being described.

La salida describe un hámster llamado Harry:

The output describes a hamster named Harry:

```
I have a hamster.  
My hamster's name is Harry.
```

Llamadas múltiples a una función (Multiple Function Calls)

Puedes llamar a una función tantas veces como sea necesario. La descripción de una segunda mascota distinta sólo requiere una llamada más a `describe_pet()` :

You can call a function as many times as needed. Describing a second, different pet requires just one more call to `describe_pet()` :

```
def describe_pet(animal_type, pet_name):  
    """Display information about a pet."""  
    print(f"\nI have a {animal_type}.")  
    print(f"My {animal_type}'s name is {pet_name.title().}")
```

```
describe_pet('hamster', 'harry')  
describe_pet('dog', 'willie')
```

En esta segunda llamada a la función, pasamos a `describe_pet()` los argumentos `'dog'` y `'willie'` . Al igual que con el anterior conjunto de argumentos que utilizamos, Python hace coincidir `'dog'` con el parámetro `animal_type` y `'willie'` con el parámetro `pet_name` . Como antes, la función hace su trabajo, pero esta vez imprime los valores de un perro llamado Willie. Ahora tenemos un hámster llamado Harry y un perro llamado Willie:

In this second function call, we pass `describe_pet()` the arguments `'dog'` and `'willie'` . As with the previous set of arguments we used, Python matches `'dog'` with the parameter `animal_type` and `'willie'` with the parameter `pet_name` . As before, the function does its job, but this time it prints values for a dog named Willie. Now we have a hamster named Harry and a dog named Willie:

```
I have a hamster.  
My hamster's name is Harry.
```

```
I have a dog.  
My dog's name is Willie.
```

Llamar a una función varias veces es una forma muy eficaz de trabajar. El código que describe una mascota se escribe una vez en la función. Luego, cada vez que quieras describir una nueva mascota, llamas a la función con la información de la nueva mascota. Aunque el código para describir una mascota se ampliara a 10 líneas, podrías describir una nueva mascota en una sola línea llamando de nuevo a la función.

Calling a function multiple times is a very efficient way to work. The code describing a pet is written once in the function. Then, anytime you want to describe a new pet, you call the function with the new pet's information. Even if the code for describing a pet were to

expand to 10 lines, you could still describe a new pet in just one line by calling the function again.

El orden importa en los argumentos posicionales (Order Matters in Positional Arguments)

Puedes obtener resultados inesperados si confundes el orden de los argumentos en una llamada a función cuando utilizas argumentos posicionales:

You can get unexpected results if you mix up the order of the arguments in a function call when using positional arguments:

```
def describe_pet(animal_type, pet_name):  
    """Display information about a pet."""  
    print(f"\nI have a {animal_type}.")  
    print(f"My {animal_type}'s name is {pet_name.title().}")  
  
describe_pet('harry', 'hamster')
```

En esta llamada a la función, enumeramos primero el nombre y después el tipo de animal. Como el argumento 'harry' aparece primero esta vez, ese valor se asigna al parámetro `animal_type`. Del mismo modo, 'hamster' se asigna a `pet_name`. Ahora tenemos un "harry" llamado "Hamster":

In this function call, we list the name first and the type of animal second. Because the argument 'harry' is listed first this time, that value is assigned to the parameter `animal_type`. Likewise, 'hamster' is assigned to `pet_name`. Now we have a "harry" named "Hamster":

```
I have a harry.  
My harry's name is Hamster.
```

Si obtienes resultados raros como éste, comprueba que el orden de los argumentos en tu llamada a la función coincide con el orden de los parámetros en la definición de la función.

If you get funny results like this, check to make sure the order of the arguments in your function call matches the order of the parameters in the function's definition.

Argumentos de palabra clave (Keyword Arguments)

Un *keyword argument* es un par nombre-valor que pasas a una función. Asocias directamente el nombre y el valor dentro del argumento, de modo que cuando pases el argumento a la función, no haya confusión (no acabarás con un harry llamado Hamster). Los argumentos con palabras clave te liberan de tener que preocuparte de ordenar correctamente tus argumentos en la llamada a la función, y aclaran el papel de cada valor en la llamada a la función.

A *keyword argument* is a name-value pair that you pass to a function. You directly associate the name and the value within the argument, so when you pass the argument to the function, there's no confusion (you won't end up with a harry named Hamster). Keyword arguments free you from having to worry about correctly ordering your arguments in the function call, and they clarify the role of each value in the function call.

Reescribamos *pets.py* utilizando argumentos de palabra clave para llamar a `describe_pet()` :

Let's rewrite *pets.py* using keyword arguments to call `describe_pet()` :

```
def describe_pet(animal_type, pet_name):  
    """Display information about a pet."""  
    print(f"\nI have a {animal_type}.")  
    print(f"My {animal_type}'s name is {pet_name.title().}")
```

```
describe_pet(animal_type='hamster', pet_name='harry')
```

La función `describe_pet()` no ha cambiado. Pero cuando llamamos a la función, le decimos explícitamente a Python con qué parámetro debe coincidir cada argumento. Cuando Python lee la llamada a la función, sabe que debe asignar el argumento 'hamster' al parámetro `animal_type` y el argumento 'harry' a `pet_name` . La salida muestra correctamente que tenemos un hámster llamado Harry.

The function `describe_pet()` hasn't changed. But when we call the function, we explicitly tell Python which parameter each argument should be matched with. When Python reads the function call, it knows to assign the argument 'hamster' to the parameter `animal_type`

and the argument 'harry' to pet_name . The output correctly shows that we have a hamster named Harry.

El orden de los argumentos de las palabras clave no importa porque Python sabe dónde debe ir cada valor. Las dos llamadas a funciones siguientes son equivalentes:

The order of keyword arguments doesn't matter because Python knows where each value should go. The following two function calls are equivalent:

```
describe_pet(animal_type='hamster', pet_name='harry')  
describe_pet(pet_name='harry', animal_type='hamster')
```

Nota (Note)

Cuando utilices argumentos de palabra clave, asegúrate de utilizar los nombres exactos de los parámetros en la definición de la función.

When you use keyword arguments, be sure to use the exact names of the parameters in the function's definition.

Valores por defecto (Default Values)

Al escribir una función, puedes definir un *default value* para cada parámetro. Si se proporciona un argumento para un parámetro en la llamada a la función, Python utiliza el valor del argumento. Si no, utiliza el valor por defecto del parámetro. Así que cuando defines un valor por defecto para un parámetro, puedes excluir el argumento correspondiente que normalmente escribirías en la llamada a la función. Utilizar valores por defecto puede simplificar tus llamadas a funciones y aclarar la forma en que se utilizan normalmente tus funciones.

When writing a function, you can define a *default value* for each parameter. If an argument for a parameter is provided in the function call, Python uses the argument value. If not, it uses the parameter's default value. So when you define a default value for a parameter, you can exclude the corresponding argument you'd usually write in the function call. Using

default values can simplify your function calls and clarify the ways your functions are typically used.

Por ejemplo, si observas que la mayoría de las llamadas a `describe_pet()` se utilizan para describir perros, puedes establecer el valor por defecto de `animal_type` en `'dog'` . Ahora cualquiera que llame a `describe_pet()` para describir un perro puede omitir esa información:

For example, if you notice that most of the calls to `describe_pet()` are being used to describe dogs, you can set the default value of `animal_type` to `'dog'` . Now anyone calling `describe_pet()` for a dog can omit that information:

```
def describe_pet(pet_name, animal_type='dog'):
    """Display information about a pet."""
    print(f"\nI have a {animal_type}.")
    print(f"My {animal_type}'s name is {pet_name.title()}")
```

```
describe_pet(pet_name='willie')
```

Cambiamos la definición de `describe_pet()` para incluir un valor por defecto, `'dog'` , para `animal_type` . Ahora, cuando se llama a la función sin especificar `animal_type` , Python sabe que debe utilizar el valor `'dog'` para este parámetro:

We changed the definition of `describe_pet()` to include a default value, `'dog'` , for `animal_type` . Now when the function is called with no `animal_type` specified, Python knows to use the value `'dog'` for this parameter:

```
I have a dog.
My dog's name is Willie.
```

Ten en cuenta que ha habido que cambiar el orden de los parámetros en la definición de la función. Como el valor por defecto hace innecesario especificar un tipo de animal como argumento, el único argumento que queda en la llamada a la función es el nombre de la mascota. Python sigue interpretándolo como un argumento posicional, por lo que si se llama a la función sólo con el nombre de la mascota, ese argumento coincidirá con el primer parámetro de la definición de la función. Esta es la razón por la que el primer parámetro debe ser `pet_name` .

Note that the order of the parameters in the function definition had to be changed. Because the default value makes it unnecessary to specify a type of animal as an argument, the only argument left in the function call is the pet's name. Python still interprets this as a positional argument, so if the function is called with just a pet's name, that argument will match up with the first parameter listed in the function's definition. This is the reason the first parameter needs to be `pet_name` .

La forma más sencilla de utilizar esta función ahora es proporcionar sólo el nombre de un perro en la llamada a la función:

The simplest way to use this function now is to provide just a dog's name in the function call:

```
describe_pet('willie')
```

Esta llamada a la función tendría el mismo resultado que el ejemplo anterior. El único argumento proporcionado es `'willie'` , por lo que se empareja con el primer parámetro de la definición, `pet_name` . Como no se proporciona ningún argumento para `animal_type` , Python utiliza el valor por defecto `'dog'` .

This function call would have the same output as the previous example. The only argument provided is `'willie'` , so it is matched up with the first parameter in the definition, `pet_name` . Because no argument is provided for `animal_type` , Python uses the default value `'dog'` .

Para describir un animal que no sea un perro, podrías utilizar una llamada a una función como ésta:

To describe an animal other than a dog, you could use a function call like this:

```
describe_pet(pet_name='harry', animal_type='hamster')
```

Como se proporciona un argumento explícito para `animal_type` , Python ignorará el valor por defecto del parámetro.

Because an explicit argument for `animal_type` is provided, Python will ignore the parameter's default value.

Nota (Note)

Cuando utilices valores por defecto, cualquier parámetro con un valor por defecto debe aparecer después de todos los parámetros que no tengan valores por defecto. Esto permite que Python siga interpretando correctamente los argumentos posicionales.

When you use default values, any parameter with a default value needs to be listed after all the parameters that don't have default values. This allows Python to continue interpreting positional arguments correctly.

Llamadas a funciones equivalentes (Equivalent Function Calls)

Como los argumentos posicionales, los argumentos de palabra clave y los valores por defecto pueden utilizarse juntos, a menudo tendrás varias formas equivalentes de llamar a una función. Considera la siguiente definición de `describe_pet()` con un valor por defecto:

Because positional arguments, keyword arguments, and default values can all be used together, you'll often have several equivalent ways to call a function. Consider the following definition for `describe_pet()` with one default value provided:

```
def describe_pet(pet_name, animal_type='dog'):
```

Con esta definición, siempre es necesario proporcionar un argumento para `pet_name` , y este valor puede proporcionarse utilizando el formato posicional o de palabra clave. Si el animal que se describe no es un perro, debe incluirse en la llamada un argumento para `animal_type` , y este argumento también puede especificarse utilizando el formato posicional o de palabra clave.

With this definition, an argument always needs to be provided for `pet_name` , and this value can be provided using the positional or keyword format. If the animal being described is not a dog, an argument for `animal_type` must be included in the call, and this argument can also be specified using the positional or keyword format.

Todas las llamadas siguientes servirían para esta función:

All of the following calls would work for this function:

```
# A dog named Willie.  
describe_pet('willie')  
describe_pet(pet_name='willie')
```

```
# A hamster named Harry.  
describe_pet('harry', 'hamster')  
describe_pet(pet_name='harry', animal_type='hamster')  
describe_pet(animal_type='hamster', pet_name='harry')
```

Cada una de estas llamadas a la función tendría la misma salida que los ejemplos anteriores.

Each of these function calls would have the same output as the previous examples.

En realidad, no importa qué estilo de llamada utilices. Mientras tus llamadas a funciones produzcan la salida que desees, utiliza el estilo que te resulte más fácil de entender.

It doesn't really matter which calling style you use. As long as your function calls produce the output you want, just use the style you find easiest to understand.

Evitar errores en los argumentos *(Avoiding Argument Errors)*

Cuando empieces a utilizar funciones, no te sorprendas si encuentras errores por argumentos no coincidentes. Los argumentos no coincidentes se producen cuando

proporcionas menos o más argumentos de los que una función necesita para hacer su trabajo. Por ejemplo, esto es lo que ocurre si intentamos llamar a `describe_pet()` sin argumentos:

When you start to use functions, don't be surprised if you encounter errors about unmatched arguments. Unmatched arguments occur when you provide fewer or more arguments than a function needs to do its work. For example, here's what happens if we try to call `describe_pet()` with no arguments:

```
def describe_pet(animal_type, pet_name):  
    """Display information about a pet."""  
    print(f"\nI have a {animal_type}.")  
    print(f"My {animal_type}'s name is {pet_name.title().}")
```

```
describe_pet()
```

Python reconoce que falta alguna información en la llamada a la función, y el rastreo nos lo indica:

Python recognizes that some information is missing from the function call, and the traceback tells us that:

Traceback (most recent call last):

❶ File "pets.py", line 6, in <module>

❷ `describe_pet()`

^^^^^^^^^^^^^^^^^^

❸ `TypeError: describe_pet() missing 2 required positional arguments: 'animal_type' and 'pet_name'`

En primer lugar, el rastreo nos indica la ubicación del problema ❶, lo que nos permite mirar atrás y ver que algo salió mal en nuestra llamada a la función. A continuación, se escribe la llamada a la función infractora para que la veamos ❷. Por último, el rastreo nos dice que a la llamada le faltan dos argumentos e informa de los nombres de los argumentos que faltan ❸. Si esta función estuviera en un archivo independiente, probablemente podríamos reescribir la llamada correctamente sin tener que abrir ese archivo y leer el código de la función.

The traceback first tells us the location of the problem ❶ , allowing us to look back and see that something went wrong in our function call. Next, the offending function call is written out for us to see ❷ . Last, the traceback tells us the call is missing two arguments and reports the names of the missing arguments ❸ . If this function were in a separate file, we could probably rewrite the call correctly without having to open that file and read the function code.

Python es útil porque lee el código de la función por nosotros y nos dice los nombres de los argumentos que tenemos que proporcionar. Esta es otra motivación para dar nombres descriptivos a tus variables y funciones. Si lo haces, los mensajes de error de Python serán más útiles para ti y para cualquiera que pueda utilizar tu código.

Python is helpful in that it reads the function's code for us and tells us the names of the arguments we need to provide. This is another motivation for giving your variables and functions descriptive names. If you do, Python's error messages will be more useful to you and anyone else who might use your code.

Si proporcionas demasiados argumentos, deberías obtener un rastreo similar que puede ayudarte a hacer coincidir correctamente tu llamada a la función con la definición de la función.

If you provide too many arguments, you should get a similar traceback that can help you correctly match your function call to the function definition.

Pruébalo tú mismo (*Try It Yourself*)

8-3. Camiseta: Escribe una función llamada `make_shirt()` que acepte una talla y el texto de un mensaje que debe imprimirse en la camiseta. La función debe imprimir una frase que resuma la talla de la camiseta y el mensaje impreso en ella.

8-3. T-Shirt: Write a function called `make_shirt()` that accepts a size and the text of a message that should be printed on the shirt. The function should print a sentence summarizing the size of the shirt and the message printed on it.

Llama a la función una vez utilizando argumentos posicionales para hacer una camisa.
Llama a la función una segunda vez utilizando argumentos de palabra clave.

Call the function once using positional arguments to make a shirt. Call the function a second time using keyword arguments.

8-4. Camisas grandes: Modifica la función `make_shirt()` para que las camisas sean grandes por defecto con un mensaje que diga *I love Python* . Haz una camisa grande y una mediana con el mensaje por defecto, y una camisa de cualquier talla con un mensaje diferente.

8-4. Large Shirts: Modify the `make_shirt()` function so that shirts are large by default with a message that reads *I love Python* . Make a large shirt and a medium shirt with the default message, and a shirt of any size with a different message.

8-5. Ciudades: Escribe una función llamada `describe_city()` que acepte el nombre de una ciudad y su país. La función debe imprimir una frase sencilla, como *Reykjavik is in Iceland* . Dale al parámetro del país un valor por defecto. Llama a tu función para tres ciudades diferentes, de las que al menos una no esté en el país por defecto.

8-5. Cities: Write a function called `describe_city()` that accepts the name of a city and its country. The function should print a simple sentence, such as *Reykjavik is in Iceland* . Give the parameter for the country a default value. Call your function for three different cities, at least one of which is not in the default country.

Valores de retorno (Return Values)

Una función no siempre tiene que mostrar su resultado directamente. En su lugar, puede procesar algunos datos y luego devolver un valor o conjunto de valores. El valor que devuelve la función se llama *return value* . La sentencia `return` toma un valor del interior de una función y lo devuelve a la línea que llamó a la función. Los valores de retorno te permiten trasladar gran parte del trabajo pesado de tu programa a las funciones, lo que puede simplificar el cuerpo de tu programa.

A function doesn't always have to display its output directly. Instead, it can process some data and then return a value or set of values. The value the function returns is called a *return value*. The `return` statement takes a value from inside a function and sends it back to the line that called the function. Return values allow you to move much of your program's grunt work into functions, which can simplify the body of your program.

Devolver un valor simple (*Returning a Simple Value*)

Veamos una función que toma un nombre y un apellido y devuelve un nombre completo formateado:

Let's look at a function that takes a first and last name, and returns a neatly formatted full name:

nombre_formateado.py

formatted_name.py

```
def get_formatted_name(first_name, last_name):
    """Return a full name, neatly formatted."""
    ❶ full_name = f'{first_name} {last_name}'
    ❷ return full_name.title()

    ❸ musician = get_formatted_name('jimi', 'hendrix')
    print(musician)
```

La definición de `get_formatted_name()` toma como parámetros un nombre y un apellido. La función combina estos dos nombres, añade un espacio entre ellos y asigna el resultado a `full_name` ❶. El valor de `full_name` se convierte a mayúsculas y minúsculas, y luego se devuelve a la línea de llamada ❷.

The definition of `get_formatted_name()` takes as parameters a first and last name. The function combines these two names, adds a space between them, and assigns the result to `full_name` ❶. The value of `full_name` is converted to title case, and then returned to the calling line ❷.

Cuando llamas a una función que devuelve un valor, tienes que proporcionar una variable a la que se pueda asignar el valor devuelto. En este caso, el valor devuelto se asigna a la variable `musician` ③ . La salida muestra un nombre bien formateado compuesto por las partes del nombre de una persona:

When you call a function that returns a value, you need to provide a variable that the return value can be assigned to. In this case, the returned value is assigned to the variable `musician` ③ . The output shows a neatly formatted name made up of the parts of a person's name:

Jimi Hendrix

Esto puede parecer mucho trabajo para obtener un nombre bien formateado cuando podríamos haber escrito simplemente:

This might seem like a lot of work to get a neatly formatted name when we could have just written:

```
print("Jimi Hendrix")
```

Sin embargo, cuando te planteas trabajar con un programa grande que necesita almacenar muchos nombres y apellidos por separado, funciones como `get_formatted_name()` resultan muy útiles. Almacenas los nombres y apellidos por separado y luego llamas a esta función siempre que quieras mostrar un nombre completo.

However, when you consider working with a large program that needs to store many first and last names separately, functions like `get_formatted_name()` become very useful. You store first and last names separately and then call this function whenever you want to display a full name.

Convertir un argumento en opcional (*Making an Argument Optional*)

A veces tiene sentido hacer que un argumento sea opcional, para que las personas que utilicen la función puedan elegir proporcionar información extra sólo si lo desean. Puedes utilizar valores por defecto para hacer que un argumento sea opcional.

Sometimes it makes sense to make an argument optional, so that people using the function can choose to provide extra information only if they want to. You can use default values to make an argument optional.

Por ejemplo, supongamos que queremos ampliar `get_formatted_name()` para que también incluya los segundos nombres. Un primer intento de incluir los segundos nombres podría tener este aspecto:

For example, say we want to expand `get_formatted_name()` to handle middle names as well. A first attempt to include middle names might look like this:

```
def get_formatted_name(first_name, middle_name, last_name):  
    """Return a full name, neatly formatted."""  
    full_name = f"{first_name} {middle_name} {last_name}"  
    return full_name.title()
```

```
musician = get_formatted_name('john', 'lee', 'hooker')  
print(musician)
```

Esta función funciona cuando se le da un nombre, un segundo nombre y un apellido. La función toma las tres partes del nombre y construye una cadena a partir de ellas. La función añade espacios donde corresponda y convierte el nombre completo a mayúsculas y minúsculas:

This function works when given a first, middle, and last name. The function takes in all three parts of a name and then builds a string out of them. The function adds spaces where appropriate and converts the full name to title case:

John Lee Hooker

Pero los segundos nombres no siempre son necesarios, y esta función, tal como está escrita, no funcionaría si intentaras llamarla sólo con un nombre y un apellido. Para que el segundo nombre sea opcional, podemos dar al argumento `middle_name` un valor por defecto vacío e ignorar el argumento a menos que el usuario proporcione un valor. Para que `get_formatted_name()` funcione sin segundo nombre, establecemos el valor por defecto de `middle_name` en una cadena vacía y lo movemos al final de la lista de parámetros:

But middle names aren't always needed, and this function as written would not work if you tried to call it with only a first name and a last name. To make the middle name optional, we can give the `middle_name` argument an empty default value and ignore the argument unless the user provides a value. To make `get_formatted_name()` work without a middle name, we set the default value of `middle_name` to an empty string and move it to the end of the list of parameters:

```
def get_formatted_name(first_name, last_name, middle_name=""):
```

```
    """Return a full name, neatly formatted."""
```

```
    ❶ if middle_name:
        full_name = f"{first_name} {middle_name} {last_name}"
    ❷ else:
        full_name = f"{first_name} {last_name}"
    return full_name.title()
```

```
musician = get_formatted_name('jimi', 'hendrix')
print(musician)
```

```
❸ musician = get_formatted_name('john', 'hooker', 'lee')
print(musician)
```

En este ejemplo, el nombre se construye a partir de tres partes posibles. Como siempre hay un nombre y un apellido, estos parámetros aparecen primero en la definición de la función. El segundo nombre es opcional, por lo que aparece el último en la definición, y su valor por defecto es una cadena vacía.

In this example, the name is built from three possible parts. Because there's always a first and last name, these parameters are listed first in the function's definition. The middle name is optional, so it's listed last in the definition, and its default value is an empty string.

En el cuerpo de la función, comprobamos si se ha proporcionado un segundo nombre. Python interpreta las cadenas no vacías como `True`, por lo que la prueba condicional `if middle_name` se evalúa como `True` si hay un argumento de segundo nombre en la llamada a la función ❶. Si se proporciona un segundo nombre, el nombre, el segundo nombre y los apellidos se combinan para formar un nombre completo. Este nombre se cambia a mayúsculas y se devuelve a la línea de llamada a la función, donde se asigna a la variable `musician` y se imprime. Si no se proporciona el segundo nombre, la cadena vacía falla la prueba `if` y el bloque `else` ejecuta ❷. El nombre completo se hace sólo con nombre y apellidos, y el nombre formateado se devuelve a la línea de llamada, donde se asigna a `musician` y se imprime.

In the body of the function, we check to see if a middle name has been provided. Python interprets non-empty strings as `True`, so the conditional test `if middle_name` evaluates to `True` if a middle name argument is in the function call ❶. If a middle name is provided, the first, middle, and last names are combined to form a full name. This name is then changed to title case and returned to the function call line, where it's assigned to the variable `musician` and printed. If no middle name is provided, the empty string fails the `if` test and the `else` block runs ❷. The full name is made with just a first and last name, and the formatted name is returned to the calling line where it's assigned to `musician` and printed.

Llamar a esta función con un nombre y apellidos es sencillo. Sin embargo, si utilizamos un segundo nombre, tenemos que asegurarnos de que el segundo nombre es el último argumento pasado para que Python empareje correctamente los argumentos posicionales ❸.

Calling this function with a first and last name is straightforward. If we're using a middle name, however, we have to make sure the middle name is the last argument passed so Python will match up the positional arguments correctly ❸.

Esta versión modificada de nuestra función funciona para las personas que sólo tienen nombre y apellidos, y también para las que tienen segundo nombre:

This modified version of our function works for people with just a first and last name, and it works for people who have a middle name as well:

Jimi Hendrix
John Lee Hooker

Los valores opcionales permiten a las funciones manejar una amplia gama de casos de uso, al tiempo que permiten que las llamadas a la función sigan siendo lo más sencillas posible.

Optional values allow functions to handle a wide range of use cases while letting function calls remain as simple as possible.

Devolver un diccionario *(Returning a Dictionary)*

Una función puede devolver cualquier tipo de valor que necesites, incluidas estructuras de datos más complicadas como listas y diccionarios. Por ejemplo, la siguiente función toma partes de un nombre y devuelve un diccionario que representa a una persona:

A function can return any kind of value you need it to, including more complicated data structures like lists and dictionaries. For example, the following function takes in parts of a name and returns a dictionary representing a person:

persona.py

person.py

```
def build_person(first_name, last_name):  
    """Return a dictionary of information about a person."""  
    ❶ person = {'first': first_name, 'last': last_name}  
    ❷ return person
```

```
musician = build_person('jimi', 'hendrix')  
❸ print(musician)
```

La función `build_person()` toma un nombre y un apellido, y pone estos valores en un diccionario ❶. El valor de `first_name` se almacena con la clave `'first'`, y el valor de `last_name` se almacena con la clave `'last'`. A continuación, se devuelve el diccionario completo que representa a la persona ❷. El valor devuelto se imprime ❸ con las dos piezas originales de información textual almacenadas ahora en un diccionario:

The function `build_person()` takes in a first and last name, and puts these values into a dictionary ❶. The value of `first_name` is stored with the key `'first'`, and the value of `last_name` is stored with the key `'last'`. Then, the entire dictionary representing the person is returned ❷. The return value is printed ❸ with the original two pieces of textual information now stored in a dictionary:

```
{'first': 'jimi', 'last': 'hendrix'}
```


Esta función toma información textual simple y la coloca en una estructura de datos más significativa que te permite trabajar con la información más allá de simplemente imprimirla. Las cadenas 'jimi' y 'hendrix' se etiquetan ahora como nombre y apellidos. Puedes ampliar fácilmente esta función para que acepte valores opcionales como un segundo nombre, una edad, una ocupación o cualquier otra información que quieras almacenar sobre una persona. Por ejemplo, el siguiente cambio te permite almacenar también la edad de una persona:

This function takes in simple textual information and puts it into a more meaningful data structure that lets you work with the information beyond just printing it. The strings 'jimi' and 'hendrix' are now labeled as a first name and last name. You can easily extend this function to accept optional values like a middle name, an age, an occupation, or any other information you want to store about a person. For example, the following change allows you to store a person's age as well:

```
def build_person(first_name, last_name, age=None):
    """Return a dictionary of information about a person."""
    person = {'first': first_name, 'last': last_name}
    if age:
        person['age'] = age
    return person

musician = build_person('jimi', 'hendrix', age=27)
print(musician)
```

Añadimos un nuevo parámetro opcional `age` a la definición de la función y asignamos al parámetro el valor especial `None`, que se utiliza cuando una variable no tiene asignado ningún valor específico. Puedes pensar en `None` como un valor marcador de posición. En las pruebas condicionales, `None` se evalúa como `False`. Si la llamada a la función incluye un valor para `age`, ese valor se almacena en el diccionario. Esta función siempre almacena el nombre de una persona, pero también puede modificarse para almacenar cualquier otra información que desees sobre una persona.

We add a new optional parameter `age` to the function definition and assign the parameter the special value `None`, which is used when a variable has no specific value assigned to it. You can think of `None` as a placeholder value. In conditional tests, `None` evaluates to `False`. If the function call includes a value for `age`, that value is stored in the dictionary. This function always stores a person's name, but it can also be modified to store any other information you want about a person.

Utilizar una función con un bucle while *(Using a Function with a while Loop)*

Puedes utilizar funciones con todas las estructuras de Python que has aprendido hasta ahora. Por ejemplo, vamos a utilizar la función `get_formatted_name()` con un bucle `while` para saludar más formalmente a los usuarios. Aquí tienes un primer intento de saludar a la gente utilizando su nombre y apellidos:

You can use functions with all the Python structures you've learned about so far. For example, let's use the `get_formatted_name()` function with a `while` loop to greet users more formally. Here's a first attempt at greeting people using their first and last names:

saludador.py

greeter.py

```
def get_formatted_name(first_name, last_name):
    """Return a full name, neatly formatted."""
    full_name = f'{first_name} {last_name}'
    return full_name.title()

# This is an infinite loop!
while True:
    ❶ print("\nPlease tell me your name:")
    f_name = input("First name: ")
    l_name = input("Last name: ")

    formatted_name = get_formatted_name(f_name, l_name)
    print(f"\nHello, {formatted_name}!")
```

Para este ejemplo, utilizamos una versión sencilla de `get_formatted_name()` que no incluye los segundos nombres. El bucle `while` pide al usuario que introduzca su nombre, y le preguntamos su nombre y apellidos por separado ❶ .

For this example, we use a simple version of `get_formatted_name()` that doesn't involve middle names. The `while` loop asks the user to enter their name, and we prompt for their first and last name separately ❶ .

Pero hay un problema con este bucle `while` : No hemos definido una condición de abandono. ¿Dónde pones una condición de abandono cuando pides una serie de entradas ? Queremos que el usuario pueda salir lo más fácilmente posible, así que cada pregunta debe ofrecer una forma de salir. La sentencia `break` ofrece una forma directa de salir del bucle en cualquiera de los dos indicadores:

But there's one problem with this `while` loop: We haven't defined a quit condition. Where do you put a quit condition when you ask for a series of inputs? We want the user to be able to quit as easily as possible, so each prompt should offer a way to quit. The `break` statement offers a straightforward way to exit the loop at either prompt:

```
def get_formatted_name(first_name, last_name):
    """Return a full name, neatly formatted."""
    full_name = f'{first_name} {last_name}'
    return full_name.title()

while True:
    print("\nPlease tell me your name:")
    print("(enter 'q' at any time to quit)")

    f_name = input("First name: ")
    if f_name == 'q':
        break

    l_name = input("Last name: ")
    if l_name == 'q':
        break

    formatted_name = get_formatted_name(f_name, l_name)
    print(f"\nHello, {formatted_name}!")
```

Añadimos un mensaje que informa al usuario de cómo salir y, a continuación, salimos del bucle si el usuario introduce el valor quit en cualquiera de los dos indicadores. Ahora el programa seguirá saludando a la gente hasta que alguien introduzca `q` para cualquiera de los dos nombres:

We add a message that informs the user how to quit, and then we break out of the loop if the user enters the quit value at either prompt. Now the program will continue greeting people until someone enters `q` for either name:

```
Please tell me your name:
(enter 'q' at any time to quit)
First name: eric
Last name: matthes
```

```
Hello, Eric Matthes!
```

```
Please tell me your name:
(enter 'q' at any time to quit)
First name: q
```

Pruébalo tú mismo *(Try It Yourself)*

8-6. Nombres de ciudades: Escribe una función llamada `city_country()` que reciba el nombre de una ciudad y su país. La función debe devolver una cadena con el siguiente formato:

8-6. City Names: Write a function called `city_country()` that takes in the name of a city and its country. The function should return a string formatted like this:

```
"Santiago, Chile"
```

Llama a tu función con al menos tres pares ciudad-país e imprime los valores devueltos.

Call your function with at least three city-country pairs, and print the values that are returned.

8-7. Álbum: Escribe una función llamada `make_album()` que construya un diccionario que describa un álbum de música. La función debe recibir un nombre de artista y un título de álbum, y debe devolver un diccionario que contenga estas dos informaciones. Utiliza la función para crear tres diccionarios que representen distintos álbumes. Imprime cada valor devuelto para mostrar que los diccionarios almacenan correctamente la información del

álbum.

8-7. Album: Write a function called `make_album()` that builds a dictionary describing a music album. The function should take in an artist name and an album title, and it should return a dictionary containing these two pieces of information. Use the function to make three dictionaries representing different albums. Print each return value to show that the dictionaries are storing the album information correctly.

Utiliza `None` para añadir un parámetro opcional a `make_album()` que te permita almacenar el número de canciones de un álbum. Si la línea de llamada incluye un valor para el número de canciones, añade ese valor al diccionario del álbum. Haz al menos una nueva llamada a una función que incluya el número de canciones de un álbum.

Use `None` to add an optional parameter to `make_album()` that allows you to store the number of songs on an album. If the calling line includes a value for the number of songs, add that value to the album's dictionary. Make at least one new function call that includes the number of songs on an album.

8-8. Álbumes de Usuario: Empieza con tu programa del Ejercicio 8-7. Escribe un bucle `while` que permita a los usuarios introducir el artista y el título de un álbum. Una vez que tengas esa información, llama a `make_album()` con la entrada del usuario e imprime el diccionario que se crea. Asegúrate de incluir un valor de salida en el bucle `while`.

8-8. User Albums: Start with your program from Exercise 8-7. Write a `while` loop that allows users to enter an album's artist and title. Once you have that information, call `make_album()` with the user's input and print the dictionary that's created. Be sure to include a quit value in the `while` loop.

Pasar una lista (*Passing a List*)

A menudo te resultará útil pasar una lista a una función, ya sea una lista de nombres, números u objetos más complejos, como diccionarios. Cuando pasas una lista a una función, ésta obtiene acceso directo al contenido de la lista. Utilicemos las funciones para que trabajar con listas sea más eficaz.

You'll often find it useful to pass a list to a function, whether it's a list of names, numbers, or more complex objects, such as dictionaries. When you pass a list to a function, the function gets direct access to the contents of the list. Let's use functions to make working with lists more efficient.

Supongamos que tenemos una lista de usuarios y queremos imprimir un saludo a cada uno. El siguiente ejemplo envía una lista de nombres a una función llamada `greet_users()` , que saluda individualmente a cada persona de la lista:

Say we have a list of users and want to print a greeting to each. The following example sends a list of names to a function called `greet_users()` , which greets each person in the list individually:

salud_usuarios.py

greet_users.py

```
def greet_users(names):
    """Print a simple greeting to each user in the list."""
    for name in names:
        msg = f'Hello, {name.title()}!'
        print(msg)
```

```
usernames = ['hannah', 'ty', 'margot']
greet_users(usernames)
```

Definimos `greet_users()` para que espere una lista de nombres, que asigna al parámetro `names` . La función recorre la lista que recibe e imprime un saludo a cada usuario. Fuera de la función, definimos una lista de usuarios y pasamos la lista `usernames` a `greet_users()` en la llamada a la función:

We define `greet_users()` so it expects a list of names, which it assigns to the parameter `names` . The function loops through the list it receives and prints a greeting to each user.

Outside of the function, we define a list of users and then pass the list `usernames` to `greet_users()` in the function call:

```
Hello, Hannah!  
Hello, Ty!  
Hello, Margot!
```

Éste es el resultado que queríamos. Cada usuario ve un saludo personalizado, y puedes llamar a la función siempre que quieras saludar a un conjunto específico de usuarios.

This is the output we wanted. Every user sees a personalized greeting, and you can call the function anytime you want to greet a specific set of users.

Modificar una lista en una función *(Modifying a List in a Function)*

Cuando pasas una lista a una función, la función puede modificar la lista. Cualquier cambio realizado en la lista dentro del cuerpo de la función es permanente, lo que te permite trabajar con eficacia incluso cuando manejas grandes cantidades de datos.

When you pass a list to a function, the function can modify the list. Any changes made to the list inside the function's body are permanent, allowing you to work efficiently even when you're dealing with large amounts of data.

Piensa en una empresa que crea modelos impresos en 3D de diseños que envían los usuarios. Los diseños que necesitan ser impresos se almacenan en una lista, y después de ser impresos se mueven a otra lista. El siguiente código hace esto sin utilizar funciones:

Consider a company that creates 3D printed models of designs that users submit. Designs that need to be printed are stored in a list, and after being printed they're moved to a separate list. The following code does this without using functions:

imprimiendo_modelos.py

printing_models.py

```
# Start with some designs that need to be printed.
unprinted_designs = ['phone case', 'robot pendant', 'dodecahedron']
completed_models = []
```

```
# Simulate printing each design, until none are left.
# Move each design to completed_models after printing.
while unprinted_designs:
    current_design = unprinted_designs.pop()
    print(f'Printing model: {current_design}')
    completed_models.append(current_design)
```

```
# Display all completed models.
print("\nThe following models have been printed:")
for completed_model in completed_models:
    print(completed_model)
```

Este programa comienza con una lista de diseños que necesitan ser impresos y una lista vacía llamada `completed_models` a la que se moverá cada diseño después de ser impreso. Mientras los diseños permanezcan en `unprinted_designs`, el bucle `while` simula la impresión de cada diseño eliminando un diseño del final de la lista, almacenándolo en `current_design`, y mostrando un mensaje de que se está imprimiendo el diseño actual. A continuación, añade el diseño a la lista de modelos completados. Cuando el bucle termina de ejecutarse, se muestra una lista de los diseños que se han impreso:

This program starts with a list of designs that need to be printed and an empty list called `completed_models` that each design will be moved to after it has been printed. As long as designs remain in `unprinted_designs`, the `while` loop simulates printing each design by removing a design from the end of the list, storing it in `current_design`, and displaying a message that the current design is being printed. It then adds the design to the list of completed models. When the loop is finished running, a list of the designs that have been printed is displayed:

```
Printing model: dodecahedron
Printing model: robot pendant
Printing model: phone case
```

```
The following models have been printed:
dodecahedron
```


robot pendant
phone case

Podemos reorganizar este código escribiendo dos funciones, cada una de las cuales realiza un trabajo específico. La mayor parte del código no cambiará; sólo lo estamos estructurando con más cuidado. La primera función se encargará de imprimir los diseños, y la segunda resumirá las impresiones realizadas:

We can reorganize this code by writing two functions, each of which does one specific job. Most of the code won't change; we're just structuring it more carefully. The first function will handle printing the designs, and the second will summarize the prints that have been made:

❶ `def print_models(unprinted_designs, completed_models):`

"""

Simulate printing each design, until none are left.

Move each design to completed_models after printing.

"""

while unprinted_designs:

 current_design = unprinted_designs.pop()

 print(f'Printing model: {current_design}')
 completed_models.append(current_design)

❷ `def show_completed_models(completed_models):`

"""Show all the models that were printed."""

print("\nThe following models have been printed:")

for completed_model in completed_models:

 print(completed_model)

`unprinted_designs = ['phone case', 'robot pendant', 'dodecahedron']`

`completed_models = []`

`print_models(unprinted_designs, completed_models)`

`show_completed_models(completed_models)`

Definimos la función `print_models()` con dos parámetros: una lista de diseños que deben imprimirse y una lista de modelos completados ❶. Dadas estas dos listas, la función simula la impresión de cada diseño vaciando la lista de diseños sin imprimir y rellenando la lista de modelos completados. A continuación, definimos la función `show_completed_models()` con un parámetro: la lista de modelos completados ❷. Dada esta lista, `show_completed_models()` muestra el nombre de cada modelo que se imprimió.

We define the function `print_models()` with two parameters: a list of designs that need to be printed and a list of completed models ❶. Given these two lists, the function simulates printing each design by emptying the list of unprinted designs and filling up the list of completed models. We then define the function `show_completed_models()` with one parameter: the list of completed models ❷. Given this list, `show_completed_models()` displays the name of each model that was printed.

Este programa tiene la misma salida que la versión sin funciones, pero el código está mucho más organizado. El código que realiza la mayor parte del trabajo se ha trasladado a dos funciones separadas, lo que facilita la comprensión de la parte principal del programa. Mira el cuerpo del programa y fíjate en lo fácil que es seguir lo que ocurre:

This program has the same output as the version without functions, but the code is much more organized. The code that does most of the work has been moved to two separate functions, which makes the main part of the program easier to understand. Look at the body of the program and notice how easily you can follow what's happening:

```
unprinted_designs = ['phone case', 'robot pendant', 'dodecahedron']
completed_models = []
```

```
print_models(unprinted_designs, completed_models)
show_completed_models(completed_models)
```

Creamos una lista de diseños sin imprimir y una lista vacía que contendrá los modelos terminados. Luego, como ya hemos definido nuestras dos funciones, todo lo que tenemos que hacer es llamarlas y pasarles los argumentos adecuados. Llamamos a `print_models()` y le pasamos las dos listas que necesita; como era de esperar, `print_models()` simula la impresión de los diseños. Luego llamamos a `show_completed_models()` y le pasamos la lista de modelos completados para que nos informe de los modelos que se han impreso. Los nombres descriptivos de las funciones permiten que otros lean este código y lo entiendan, incluso sin comentarios.

We set up a list of unprinted designs and an empty list that will hold the completed models. Then, because we've already defined our two functions, all we have to do is call them and pass them the right arguments. We call `print_models()` and pass it the two lists it needs; as expected, `print_models()` simulates printing the designs. Then we call

`show_completed_models()` and pass it the list of completed models so it can report the models that have been printed. The descriptive function names allow others to read this code and understand it, even without comments.

Este programa es más fácil de ampliar y mantener que la versión sin funciones. Si más adelante necesitamos imprimir más diseños, basta con volver a llamar a `print_models()`. Si nos damos cuenta de que hay que modificar el código de impresión, podemos cambiar el código una vez, y nuestros cambios tendrán lugar en todos los lugares donde se llame a la función. Esta técnica es más eficaz que tener que actualizar el código por separado en varios lugares del programa.

This program is easier to extend and maintain than the version without functions. If we need to print more designs later on, we can simply call `print_models()` again. If we realize the printing code needs to be modified, we can change the code once, and our changes will take place everywhere the function is called. This technique is more efficient than having to update code separately in several places in the program.

Este ejemplo también demuestra la idea de que cada función debe tener un trabajo específico. La primera función imprime cada diseño, y la segunda muestra los modelos completados. Esto es más beneficioso que utilizar una función para hacer ambos trabajos. Si estás escribiendo una función y te das cuenta de que realiza demasiadas tareas diferentes, intenta dividir el código en dos funciones. Recuerda que siempre puedes llamar a una función desde otra función, lo que puede ser útil al dividir una tarea compleja en una serie de pasos.

This example also demonstrates the idea that every function should have one specific job. The first function prints each design, and the second displays the completed models. This is more beneficial than using one function to do both jobs. If you're writing a function and notice the function is doing too many different tasks, try to split the code into two functions. Remember that you can always call a function from another function, which can be helpful when splitting a complex task into a series of steps.

Evitar que una función modifique una lista *(Preventing a Function from Modifying a List)*

A veces querrás evitar que una función modifique una lista. Por ejemplo, supongamos que empiezas con una lista de diseños sin imprimir y escribes una función para moverlos a una lista de modelos terminados, como en el ejemplo anterior. Puede que decidas que, aunque hayas impreso todos los diseños, quieres conservar la lista original de diseños sin imprimir

para tus archivos. Pero como has movido todos los nombres de los diseños fuera de `unprinted_designs` , la lista está ahora vacía, y la lista vacía es la única versión que tienes; la original ha desaparecido. En este caso, puedes solucionar este problema pasando a la función una copia de la lista, no el original. Cualquier cambio que la función haga en la lista afectará sólo a la copia, dejando intacta la lista original.

Sometimes you'll want to prevent a function from modifying a list. For example, say that you start with a list of unprinted designs and write a function to move them to a list of completed models, as in the previous example. You may decide that even though you've printed all the designs, you want to keep the original list of unprinted designs for your records. But because you moved all the design names out of `unprinted_designs` , the list is now empty, and the empty list is the only version you have; the original is gone. In this case, you can address this issue by passing the function a copy of the list, not the original. Any changes the function makes to the list will affect only the copy, leaving the original list intact.

Puedes enviar una copia de una lista a una función de esta forma:

You can send a copy of a list to a function like this:

```
function_name(list_name[:])
```

La notación de corte `[:]` hace una copia de la lista para enviarla a la función. Si no quisiéramos vaciar la lista de diseños no impresos en *printing_models.py* , podríamos llamar a `print_models()` de esta forma:

The slice notation `[:]` makes a copy of the list to send to the function. If we didn't want to empty the list of unprinted designs in *printing_models.py* , we could call `print_models()` like this:

```
print_models(unprinted_designs[:], completed_models)
```

La función `print_models()` puede hacer su trabajo porque sigue recibiendo los nombres de todos los diseños no impresos. Pero esta vez utiliza una copia de la lista original de diseños sin imprimir, no la lista real `unprinted_designs` . La lista `completed_models` se llenará con los nombres de los modelos impresos como antes, pero la lista original de diseños no

impresos no se verá afectada por la función.

The function `print_models()` can do its work because it still receives the names of all unprinted designs. But this time it uses a copy of the original unprinted designs list, not the actual `unprinted_designs` list. The list `completed_models` will fill up with the names of printed models like it did before, but the original list of unprinted designs will be unaffected by the function.

Aunque puedes conservar el contenido de una lista pasando una copia de ella a tus funciones, debes pasar la lista original a las funciones a menos que tengas una razón específica para pasar una copia. Es más eficaz que una función trabaje con una lista existente, porque así se evita emplear el tiempo y la memoria necesarios para hacer una copia por separado. Esto es especialmente cierto cuando se trabaja con listas grandes.

Even though you can preserve the contents of a list by passing a copy of it to your functions, you should pass the original list to functions unless you have a specific reason to pass a copy. It's more efficient for a function to work with an existing list, because this avoids using the time and memory needed to make a separate copy. This is especially true when working with large lists.

Pruébalo tú mismo (*Try It Yourself*)

8-9. Mensajes: Haz una lista que contenga una serie de mensajes cortos de texto. Pasa la lista a una función llamada `show_messages()`, que imprime cada mensaje de texto.

8-9. Messages: Make a list containing a series of short text messages. Pass the list to a function called `show_messages()`, which prints each text message.

8-10. Enviar mensajes: Empieza con una copia de tu programa del Ejercicio 8-9. Escribe una función llamada `send_messages()` que imprima cada mensaje de texto y mueva cada mensaje a una nueva lista llamada `sent_messages` a medida que se imprime. Después de llamar a la función, imprime ambas listas para asegurarte de que los mensajes se han movido correctamente.

8-10. Sending Messages: Start with a copy of your program from Exercise 8-9. Write a function called `send_messages()` that prints each text message and moves each message to a new list called `sent_messages` as it's printed. After calling the function, print both of your lists to make sure the messages were moved correctly.

8-11. Mensajes Archivados: Empieza con tu trabajo del Ejercicio 8-10. Llama a la función `send_messages()` con una copia de la lista de mensajes. Después de llamar a la función, imprime tus dos listas para mostrar que la lista original ha conservado sus mensajes.

8-11. Archived Messages: Start with your work from Exercise 8-10. Call the function `send_messages()` with a copy of the list of messages. After calling the function, print both of your lists to show that the original list has retained its messages.

Pasar un número arbitrario de argumentos (*Passing an Arbitrary Number of Arguments*)

A veces no sabrás de antemano cuántos argumentos debe aceptar una función. Afortunadamente, Python permite que una función recoja un número arbitrario de argumentos de la sentencia que la llama.

Sometimes you won't know ahead of time how many arguments a function needs to accept. Fortunately, Python allows a function to collect an arbitrary number of arguments from the calling statement.

Por ejemplo, considera una función que construye una pizza. Necesita aceptar un número de ingredientes, pero no puedes saber de antemano cuántos ingredientes querrá una persona. La función del siguiente ejemplo tiene un parámetro, `*toppings`, pero este parámetro recoge tantos argumentos como proporcione la línea de llamada:

For example, consider a function that builds a pizza. It needs to accept a number of toppings, but you can't know ahead of time how many toppings a person will want. The function in the following example has one parameter, `*toppings`, but this parameter collects as many arguments as the calling line provides:

pizza.py

pizza.py

```
def make_pizza(*toppings):  
    """Print the list of toppings that have been requested."""  
    print(toppings)
```

```
make_pizza('pepperoni')  
make_pizza('mushrooms', 'green peppers', 'extra cheese')
```

El asterisco en el nombre del parámetro `*toppings` indica a Python que cree una tupla llamada `toppings`, que contiene todos los valores que recibe esta función. La llamada `print()` en el cuerpo de la función produce una salida que muestra que Python puede manejar una llamada a la función con un valor y una llamada con tres valores. Trata las distintas llamadas de forma similar. Observa que Python empaqueta los argumentos en una tupla, aunque la función sólo reciba un valor:

The asterisk in the parameter name `*toppings` tells Python to make a tuple called `toppings`, containing all the values this function receives. The `print()` call in the function body produces output showing that Python can handle a function call with one value and a call with three values. It treats the different calls similarly. Note that Python packs the arguments into a tuple, even if the function receives only one value:

```
('pepperoni',)  
('mushrooms', 'green peppers', 'extra cheese')
```

Ahora podemos sustituir la llamada a `print()` por un bucle que recorra la lista de ingredientes y describa la pizza que se pide:

Now we can replace the `print()` call with a loop that runs through the list of toppings and describes the pizza being ordered:

```
def make_pizza(*toppings):  
    """Summarize the pizza we are about to make."""  
    print("\nMaking a pizza with the following toppings:")
```

```
for topping in toppings:  
    print(f'- {topping}')
```

```
make_pizza('pepperoni')  
make_pizza('mushrooms', 'green peppers', 'extra cheese')
```

La función responde adecuadamente, tanto si recibe un valor como tres:

The function responds appropriately, whether it receives one value or three values:

Making a pizza with the following toppings:
- pepperoni

Making a pizza with the following toppings:
- mushrooms
- green peppers
- extra cheese

Esta sintaxis funciona independientemente del número de argumentos que reciba la función.

This syntax works no matter how many arguments the function receives.

Mezclar argumentos posicionales y arbitrarios (*Mixing Positional and Arbitrary Arguments*)

Si quieres que una función acepte varios tipos diferentes de argumentos, el parámetro que acepta un número arbitrario de argumentos debe colocarse en último lugar en la definición de la función. Python combina primero los argumentos posicionales y los de palabra clave, y luego reúne los argumentos restantes en el último parámetro.

If you want a function to accept several different kinds of arguments, the parameter that accepts an arbitrary number of arguments must be placed last in the function definition. Python matches positional and keyword arguments first and then collects any remaining arguments in the final parameter.

Por ejemplo, si la función necesita recoger un tamaño para la pizza, ese parámetro debe ir antes que el parámetro `*toppings` :

For example, if the function needs to take in a size for the pizza, that parameter must come before the parameter `*toppings` :

```
def make_pizza(size, *toppings):
    """Summarize the pizza we are about to make."""
    print(f"\nMaking a {size}-inch pizza with the following toppings:")
    for topping in toppings:
        print(f"- {topping}")

make_pizza(16, 'pepperoni')
make_pizza(12, 'mushrooms', 'green peppers', 'extra cheese')
```

En la definición de la función, Python asigna el primer valor que recibe al parámetro `size` . Todos los demás valores que vengan después se almacenan en la tupla `toppings` . Las llamadas a la función incluyen primero un argumento para el tamaño, seguido de tantos ingredientes como sean necesarios.

In the function definition, Python assigns the first value it receives to the parameter `size` . All other values that come after are stored in the tuple `toppings` . The function calls include an argument for the size first, followed by as many toppings as needed.

Ahora cada pizza tiene un tamaño y un número de ingredientes, y cada información se imprime en el lugar adecuado, mostrando primero el tamaño y después los ingredientes:

Now each pizza has a size and a number of toppings, and each piece of information is printed in the proper place, showing size first and toppings after:

Making a 16-inch pizza with the following toppings:
- pepperoni

Making a 12-inch pizza with the following toppings:
- mushrooms
- green peppers
- extra cheese

Nota (Note)

A menudo verás el nombre genérico de parámetro `*args` , que recoge argumentos posicionales arbitrarios como éste.

You'll often see the generic parameter name `*args` , which collects arbitrary positional arguments like this.

Uso de argumentos arbitrarios de palabras clave (Using Arbitrary Keyword Arguments)

A veces querrás aceptar un número arbitrario de argumentos, pero no sabrás de antemano qué tipo de información se pasará a la función. En este caso, puedes escribir funciones que acepten tantos pares clave-valor como proporcione la sentencia de llamada. Un ejemplo tiene que ver con la creación de perfiles de usuario: sabes que obtendrás información sobre un usuario, pero no estás seguro de qué tipo de información recibirás. La función `build_profile()` del siguiente ejemplo siempre recibe un nombre y un apellido, pero también acepta un número arbitrario de argumentos de palabra clave:

Sometimes you'll want to accept an arbitrary number of arguments, but you won't know ahead of time what kind of information will be passed to the function. In this case, you can write functions that accept as many key-value pairs as the calling statement provides. One example involves building user profiles: you know you'll get information about a user, but you're not sure what kind of information you'll receive. The function `build_profile()` in the following example always takes in a first and last name, but it accepts an arbitrary number of keyword arguments as well:

perfil_usuario.py

user_profile.py

```
def build_profile(first, last, **user_info):  
    """Build a dictionary containing everything we know about a user."""  
    ❶ user_info['first_name'] = first
```

```
    user_info['last_name'] = last
    return user_info

user_profile = build_profile('albert', 'einstein',
                             location='princeton',
                             field='physics')
print(user_profile)
```

La definición de `build_profile()` espera un nombre y un apellido, y luego permite al usuario pasar tantos pares nombre-valor como desee. Los asteriscos dobles delante del parámetro `**user_info` hacen que Python cree un diccionario llamado `user_info` que contiene todos los pares nombre-valor adicionales que recibe la función. Dentro de la función, puedes acceder a los pares clave-valor de `user_info` como lo harías con cualquier diccionario.

The definition of `build_profile()` expects a first and last name, and then it allows the user to pass in as many name-value pairs as they want. The double asterisks before the parameter `**user_info` cause Python to create a dictionary called `user_info` containing all the extra name-value pairs the function receives. Within the function, you can access the key-value pairs in `user_info` just as you would for any dictionary.

En el cuerpo de `build_profile()`, añadimos el nombre y los apellidos al diccionario `user_info` porque siempre recibiremos estos dos datos del usuario ❶, y aún no se han colocado en el diccionario. Después devolvemos el diccionario `user_info` a la línea de llamada a la función.

In the body of `build_profile()`, we add the first and last names to the `user_info` dictionary because we'll always receive these two pieces of information from the user ❶, and they haven't been placed into the dictionary yet. Then we return the `user_info` dictionary to the function call line.

Llamamos a `build_profile()`, pasándole el nombre 'albert', el apellido 'einstein', y los dos pares clave-valor `location='princeton'` y `field='physics'`. Asignamos el `profile` devuelto a `user_profile` e imprimimos `user_profile`:

We call `build_profile()`, passing it the first name 'albert', the last name 'einstein', and the two key-value pairs `location='princeton'` and `field='physics'`. We assign the returned profile to `user_profile` and print `user_profile`:

```
{'location': 'princeton', 'field': 'physics',  
'first_name': 'albert', 'last_name': 'einstein'}
```

El diccionario devuelto contiene el nombre y apellidos del usuario y, en este caso, también la localidad y la especialidad. La función funcionará independientemente de cuántos pares clave-valor adicionales se proporcionen en la llamada a la función.

The returned dictionary contains the user's first and last names and, in this case, the location and field of study as well. The function will work no matter how many additional key-value pairs are provided in the function call.

Puedes mezclar valores posicionales, de palabra clave y arbitrarios de muchas formas distintas al escribir tus propias funciones. Es útil saber que existen todos estos tipos de argumentos porque los verás a menudo cuando empieces a leer el código de otras personas. Utilizar correctamente los distintos tipos y saber cuándo utilizar cada uno de ellos requiere práctica. Por ahora, recuerda utilizar el método más sencillo que haga el trabajo. A medida que avances, aprenderás a utilizar cada vez el enfoque más eficaz.

You can mix positional, keyword, and arbitrary values in many different ways when writing your own functions. It's useful to know that all these argument types exist because you'll see them often when you start reading other people's code. It takes practice to use the different types correctly and to know when to use each type. For now, remember to use the simplest approach that gets the job done. As you progress, you'll learn to use the most efficient approach each time.

Nota (Note)

A menudo verás que el nombre de parámetro `**kwargs` se utiliza para recoger argumentos no específicos de palabras clave.

You'll often see the parameter name `**kwargs` used to collect nonspecific keyword arguments.

Pruébalo tú mismo (*Try It Yourself*)

8-12. Bocadillos: Escribe una función que acepte una lista de elementos que una persona quiere en un bocadillo. La función debe tener un parámetro que recoja tantos elementos como proporcione la llamada a la función, y debe imprimir un resumen del bocadillo que se está pidiendo. Llama a la función tres veces, utilizando un número distinto de argumentos cada vez.

8-12. Sandwiches: Write a function that accepts a list of items a person wants on a sandwich. The function should have one parameter that collects as many items as the function call provides, and it should print a summary of the sandwich that's being ordered. Call the function three times, using a different number of arguments each time.

8-13. Perfil de usuario: Empieza con una copia de *user_profile.py* de la página 148 . Construye un perfil tuyo llamando a `build_profile()` , utilizando tu nombre y apellidos y otros tres pares clave-valor que te describan.

8-13. User Profile: Start with a copy of *user_profile.py* from page 148 . Build a profile of yourself by calling `build_profile()` , using your first and last names and three other key-value pairs that describe you.

8-14. Coches: Escribe una función que almacene información sobre un coche en un diccionario. La función debe recibir siempre un fabricante y un nombre de modelo. A continuación, debe aceptar un número arbitrario de argumentos de palabra clave. Llama a la función con la información requerida y otros dos pares nombre-valor, como un color o una característica opcional. Tu función debería funcionar para una llamada como ésta:

8-14. Cars: Write a function that stores information about a car in a dictionary. The function should always receive a manufacturer and a model name. It should then accept an arbitrary number of keyword arguments. Call the function with the required information and two other name-value pairs, such as a color or an optional feature. Your function should work for a call like this one:

```
car = make_car('subaru', 'outback', color='blue', tow_package=True)
```

Imprime el diccionario devuelto para asegurarte de que toda la información se ha almacenado correctamente.

Print the dictionary that's returned to make sure all the information was stored correctly.

Almacenar tus funciones en módulos (*Storing Your Functions in Modules*)

Una ventaja de las funciones es la forma en que separan bloques de código de tu programa principal. Cuando utilizas nombres descriptivos para tus funciones, tus programas son mucho más fáciles de seguir. Puedes ir un paso más allá almacenando tus funciones en un archivo separado llamado *module* y luego *importing* ese módulo en tu programa principal. Una sentencia `import` indica a Python que haga que el código de un módulo esté disponible en el archivo del programa que se esté ejecutando en ese momento.

One advantage of functions is the way they separate blocks of code from your main program. When you use descriptive names for your functions, your programs become much easier to follow. You can go a step further by storing your functions in a separate file called a *module* and then *importing* that module into your main program. An `import` statement tells Python to make the code in a module available in the currently running program file.

Almacenar tus funciones en un archivo separado te permite ocultar los detalles del código de tu programa y centrarte en su lógica de alto nivel. También te permite reutilizar funciones en muchos programas diferentes. Cuando almacenas tus funciones en archivos separados, puedes compartir esos archivos con otros programadores sin que tenga que compartir todo tu programa. Saber importar funciones también te permite utilizar bibliotecas de funciones que han escrito otros programadores.

Storing your functions in a separate file allows you to hide the details of your program's code and focus on its higher-level logic. It also allows you to reuse functions in many different programs. When you store your functions in separate files, you can share those files with other programmers without having to share your entire program. Knowing how to import functions also allows you to use libraries of functions that other programmers have written.

Hay varias formas de importar un módulo, y te mostraré brevemente cada una de ellas.

There are several ways to import a module, and I'll show you each of these briefly.

Importar un módulo entero (*Importing an Entire Module*)

Para empezar a importar funciones, primero tenemos que crear un módulo. Un *module* es un archivo que termina en *.py* y que contiene el código que quieres importar a tu programa. Vamos a crear un módulo que contenga la función `make_pizza()`. Para crear este módulo, eliminaremos todo del archivo *pizza.py* excepto la función `make_pizza()` :

To start importing functions, we first need to create a module. A *module* is a file ending in *.py* that contains the code you want to import into your program. Let's make a module that contains the function `make_pizza()`. To make this module, we'll remove everything from the file *pizza.py* except the function `make_pizza()` :

pizza.py

pizza.py

```
def make_pizza(size, *toppings):
    """Summarize the pizza we are about to make."""
    print(f"\nMaking a {size}-inch pizza with the following toppings:")
    for topping in toppings:
        print(f'- {topping}')
```

Ahora crearemos un archivo independiente llamado *making_pizzas.py* en el mismo directorio que *pizza.py*. Este archivo importa el módulo que acabamos de crear y luego hace dos llamadas a `make_pizza()` :

Now we'll make a separate file called *making_pizzas.py* in the same directory as *pizza.py*. This file imports the module we just created and then makes two calls to `make_pizza()` :

haciendo_pizzas.py

making_pizzas.py

```
import pizza
```

```
❶ pizza.make_pizza(16, 'pepperoni')  
pizza.make_pizza(12, 'mushrooms', 'green peppers', 'extra cheese')
```

Cuando Python lee este archivo, la línea `import pizza` indica a Python que abra el archivo *pizza.py* y copie todas las funciones del mismo en este programa. En realidad, no ves que se copie código entre archivos, porque Python copia el código entre bastidores, justo antes de que se ejecute el programa. Todo lo que necesitas saber es que cualquier función definida en *pizza.py* estará ahora disponible en *making_pizzas.py*.

When Python reads this file, the line `import pizza` tells Python to open the file *pizza.py* and copy all the functions from it into this program. You don't actually see code being copied between files because Python copies the code behind the scenes, just before the program runs. All you need to know is that any function defined in *pizza.py* will now be available in *making_pizzas.py*.

Para llamar a una función desde un módulo importado, introduce el nombre del módulo que has importado, `pizza`, seguido del nombre de la función, `make_pizza()`, separados por un punto ❶. Este código produce la misma salida que el programa original que no importó ningún módulo:

To call a function from an imported module, enter the name of the module you imported, `pizza`, followed by the name of the function, `make_pizza()`, separated by a dot ❶. This code produces the same output as the original program that didn't import a module:

Making a 16-inch pizza with the following toppings:
- pepperoni

Making a 12-inch pizza with the following toppings:
- mushrooms

- green peppers
- extra cheese

Este primer enfoque de la importación, en el que simplemente escribes `import` seguido del nombre del módulo, hace que todas las funciones del módulo estén disponibles en tu programa. Si utilizas este tipo de sentencia `import` para importar un módulo entero llamado *module_name.py*, cada función del módulo estará disponible mediante la siguiente sintaxis:

This first approach to importing, in which you simply write `import` followed by the name of the module, makes every function from the module available in your program. If you use this kind of `import` statement to import an entire module named *module_name.py*, each function in the module is available through the following syntax:

```
module_name.function_name()
```

Importar funciones específicas (*Importing Specific Functions*)

También puedes importar una función específica de un módulo. Esta es la sintaxis general para este enfoque:

You can also import a specific function from a module. Here's the general syntax for this approach:

```
from module_name import function_name
```

Puedes importar tantas funciones como quieras de un módulo separando el nombre de cada función con una coma:

You can import as many functions as you want from a module by separating each function's name with a comma:

```
from module_name import function_0, function_1, function_2
```

El ejemplo de *making_pizzas.py* tendría este aspecto si quisiéramos importar sólo la función que vamos a utilizar:

The *making_pizzas.py* example would look like this if we want to import just the function we're going to use:

```
from pizza import make_pizza
```

```
make_pizza(16, 'pepperoni')  
make_pizza(12, 'mushrooms', 'green peppers', 'extra cheese')
```

Con esta sintaxis, no necesitas utilizar la notación de punto cuando llames a una función. Como hemos importado explícitamente la función `make_pizza()` en la sentencia `import`, podemos llamarla por su nombre cuando utilicemos la función.

With this syntax, you don't need to use the dot notation when you call a function. Because we've explicitly imported the function `make_pizza()` in the `import` statement, we can call it by name when we use the function.

Utilizar *as* para dar un alias a una función (*Using as to Give a Function an Alias*)

Si el nombre de una función que estás importando puede entrar en conflicto con un nombre existente en tu programa, o si el nombre de la función es largo, puedes utilizar un *alias* corto y único, un nombre alternativo similar a un apodo para la función. Le darás a la función este apodo especial cuando importes la función.

If the name of a function you're importing might conflict with an existing name in your program, or if the function name is long, you can use a short, unique *alias* —an alternate name similar to a nickname for the function. You'll give the function this special nickname when you import the function.

Aquí damos a la función `make_pizza()` un alias, `mp()`, importando `make_pizza` as `mp`. La palabra clave `as` cambia el nombre de una función utilizando el alias que le proporcionas:

Here we give the function `make_pizza()` an alias, `mp()`, by importing `make_pizza` as `mp`. The `as` keyword renames a function using the alias you provide:

```
from pizza import make_pizza as mp
```

```
mp(16, 'pepperoni')  
mp(12, 'mushrooms', 'green peppers', 'extra cheese')
```

La sentencia `import` mostrada aquí cambia el nombre de la función `make_pizza()` a `mp()` en este programa. Cada vez que queramos llamar a `make_pizza()` podemos simplemente escribir `mp()` en su lugar, y Python ejecutará el código en `make_pizza()` evitando cualquier confusión con otra función `make_pizza()` que puedas haber escrito en este archivo de programa.

The `import` statement shown here renames the function `make_pizza()` to `mp()` in this program. Anytime we want to call `make_pizza()` we can simply write `mp()` instead, and Python will run the code in `make_pizza()` while avoiding any confusion with another `make_pizza()` function you might have written in this program file.

La sintaxis general para proporcionar un alias es:

The general syntax for providing an alias is:

```
from module_name import function_name as fn
```

Usar `as` para dar un alias a un módulo *(Using `as` to Give a Module an Alias)*

También puedes proporcionar un alias para el nombre de un módulo. Dar a un módulo un alias corto, como `p` para `pizza`, te permite llamar a las funciones del módulo más rápidamente. Llamar a `p.make_pizza()` es más conciso que llamar a `pizza.make_pizza()` :

You can also provide an alias for a module name. Giving a module a short alias, like `p` for `pizza`, allows you to call the module's functions more quickly. Calling `p.make_pizza()` is more concise than calling `pizza.make_pizza()` :

```
import pizza as p
```

```
p.make_pizza(16, 'pepperoni')  
p.make_pizza(12, 'mushrooms', 'green peppers', 'extra cheese')
```

El módulo `pizza` recibe el alias `p` en la sentencia `import`, pero todas las funciones del módulo conservan sus nombres originales. Llamar a las funciones escribiendo `p.make_pizza()` no sólo es más conciso que `pizza.make_pizza()`, sino que también desvía tu atención del nombre del módulo y te permite centrarte en los nombres descriptivos de sus funciones. Estos nombres de función, que te indican claramente lo que hace cada función, son más importantes para la legibilidad de tu código que utilizar el nombre completo del módulo.

The module `pizza` is given the alias `p` in the `import` statement, but all of the module's functions retain their original names. Calling the functions by writing `p.make_pizza()` is not only more concise than `pizza.make_pizza()`, but it also redirects your attention from the module name and allows you to focus on the descriptive names of its functions. These function names, which clearly tell you what each function does, are more important to the readability of your code than using the full module name.

La sintaxis general de este enfoque es

The general syntax for this approach is:

```
import module_name as mn
```

Importar todas las funciones de un módulo (*Importing All Functions in a Module*)

Puedes indicar a Python que importe todas las funciones de un módulo utilizando el operador asterisco (`*`):

You can tell Python to import every function in a module by using the asterisk (`*`) operator:

```
from pizza import *
```

```
make_pizza(16, 'pepperoni')  
make_pizza(12, 'mushrooms', 'green peppers', 'extra cheese')
```

El asterisco en la sentencia `import` indica a Python que copie todas las funciones del módulo `pizza` en este archivo de programa. Como se importan todas las funciones, puedes llamar a cada una de ellas por su nombre sin utilizar la notación de punto. Sin embargo, es

mejor no utilizar este método cuando trabajes con módulos más grandes que no hayas escrito tú: si el módulo tiene un nombre de función que coincide con un nombre existente en tu proyecto, puedes obtener resultados inesperados. Python puede ver varias funciones o variables con el mismo nombre, y en lugar de importar todas las funciones por separado, las sobrescribirá.

The asterisk in the `import` statement tells Python to copy every function from the module `pizza` into this program file. Because every function is imported, you can call each function by name without using the dot notation. However, it's best not to use this approach when you're working with larger modules that you didn't write: if the module has a function name that matches an existing name in your project, you can get unexpected results. Python may see several functions or variables with the same name, and instead of importing all the functions separately, it will overwrite the functions.

Lo mejor es importar la función o funciones que quieras, o importar todo el módulo y utilizar la notación por puntos. Esto conduce a un código claro que es fácil de leer y comprender. Incluyo esta sección para que reconozcas declaraciones `import` como las siguientes cuando las veas en el código de otras personas:

The best approach is to import the function or functions you want, or import the entire module and use the dot notation. This leads to clear code that's easy to read and understand. I include this section so you'll recognize `import` statements like the following when you see them in other people's code:

```
from module_name import *
```

Funciones de estilo (*Styling Functions*)

Debes tener en cuenta algunos detalles a la hora de dar estilo a las funciones. Las funciones deben tener nombres descriptivos, y estos nombres deben utilizar letras minúsculas y guiones bajos. Los nombres descriptivos te ayudan a ti y a los demás a entender lo que intenta hacer tu código. Los nombres de los módulos también deben utilizar estas convenciones.

You need to keep a few details in mind when you're styling functions. Functions should have descriptive names, and these names should use lowercase letters and underscores. Descriptive names help you and others understand what your code is trying to do. Module names should use these conventions as well.

Cada función debe tener un comentario que explique de forma concisa lo que hace la función. Este comentario debe aparecer inmediatamente después de la definición de la función y utilizar el formato docstring. En una función bien documentada, otros programadores pueden utilizar la función leyendo únicamente la descripción del docstring. Deberían poder confiar en que el código funciona como se describe, y siempre que conozcan el nombre de la función, los argumentos que necesita y el tipo de valor que devuelve, deberían poder utilizarla en sus programas.

Every function should have a comment that explains concisely what the function does. This comment should appear immediately after the function definition and use the docstring format. In a well-documented function, other programmers can use the function by reading only the description in the docstring. They should be able to trust that the code works as described, and as long as they know the name of the function, the arguments it needs, and the kind of value it returns, they should be able to use it in their programs.

Si especificas un valor por defecto para un parámetro, no deben utilizarse espacios a ambos lados del signo igual:

If you specify a default value for a parameter, no spaces should be used on either side of the equal sign:

```
def function_name(parameter_0, parameter_1='default value')
```

La misma convención debe utilizarse para los argumentos de palabras clave en las llamadas a funciones:

The same convention should be used for keyword arguments in function calls:

```
function_name(value_0, parameter_1='value')
```

PEP 8 (<https://www.python.org/dev/peps/pep-0008>) recomienda limitar las líneas de código a 79 caracteres para que cada línea sea visible en una ventana del editor de tamaño razonable. Si un conjunto de parámetros hace que la definición de una función tenga más de 79 caracteres, pulsa INTRO después del paréntesis de apertura en la línea de definición. En la línea siguiente, pulsa dos veces la tecla TAB para separar la lista de argumentos del

cuerpo de la función, que sólo se sangrará un nivel.

PEP 8 (<https://www.python.org/dev/peps/pep-0008>) recommends that you limit lines of code to 79 characters so every line is visible in a reasonably sized editor window. If a set of parameters causes a function's definition to be longer than 79 characters, press ENTER after the opening parenthesis on the definition line. On the next line, press the TAB key twice to separate the list of arguments from the body of the function, which will only be indented one level.

La mayoría de los editores alinean automáticamente las líneas adicionales de argumentos para que coincidan con la sangría que has establecido en la primera línea:

Most editors automatically line up any additional lines of arguments to match the indentation you have established on the first line:

```
def function_name(  
    parameter_0, parameter_1, parameter_2,  
    parameter_3, parameter_4, parameter_5):  
    function body...
```

Si tu programa o módulo tiene más de una función, puedes separar cada una de ellas con dos líneas en blanco para que sea más fácil ver dónde acaba una función y empieza la siguiente.

If your program or module has more than one function, you can separate each by two blank lines to make it easier to see where one function ends and the next one begins.

Todas las declaraciones `import` deben escribirse al principio de un archivo. La única excepción es si utilizas comentarios al principio de tu archivo para describir el programa en general.

All `import` statements should be written at the beginning of a file. The only exception is if you use comments at the beginning of your file to describe the overall program.

Pruébalo tú mismo (Try It Yourself)

8-15. Impresión de modelos: Coloca las funciones del ejemplo *printing_models.py* en un archivo aparte llamado *printing_functions.py* . Escribe una sentencia `import` al principio de *printing_models.py* , y modifica el archivo para utilizar las funciones importadas.

8-15. Printing Models: Put the functions for the example *printing_models.py* in a separate file called *printing_functions.py* . Write an import statement at the top of *printing_models.py* , and modify the file to use the imported functions.

8-16. Importaciones: Utilizando un programa que hayas escrito y que contenga una función, guarda esa función en un archivo aparte. Importa la función a tu archivo de programa principal, y llama a la función utilizando cada uno de estos enfoques:

8-16. Imports: Using a program you wrote that has one function in it, store that function in a separate file. Import the function into your main program file, and call the function using each of these approaches:

```
import module_name
from module_name import function_name
from module_name import function_name as fn
import module_name as mn
from module_name import *
```

8-17. Estilizar funciones: Elige tres programas cualesquiera que hayas escrito para este capítulo, y asegúrate de que siguen las pautas de estilo descritas en esta sección.

8-17. Styling Functions: Choose any three programs you wrote for this chapter, and make sure they follow the styling guidelines described in this section.

Resumen (Summary)

En este capítulo has aprendido a escribir funciones y a pasar argumentos para que tus funciones tengan acceso a la información que necesitan para hacer su trabajo. Aprendiste a utilizar argumentos posicionales y de palabra clave, y también a aceptar un número arbitrario de argumentos. Viste funciones que muestran la salida y funciones que devuelven valores. Aprendiste a utilizar funciones con listas, diccionarios, sentencias `if` y bucles `while`. También aprendiste a almacenar tus funciones en archivos separados llamados *modules*, para que los archivos de tu programa sean más sencillos y fáciles de entender. Por último, aprendiste a dar estilo a tus funciones para que tus programas sigan estando bien estructurados y sean lo más fáciles de leer para ti y para los demás.

In this chapter, you learned how to write functions and to pass arguments so that your functions have access to the information they need to do their work. You learned how to use positional and keyword arguments, and also how to accept an arbitrary number of arguments. You saw functions that display output and functions that return values. You learned how to use functions with lists, dictionaries, `if` statements, and `while` loops. You also saw how to store your functions in separate files called *modules*, so your program files will be simpler and easier to understand. Finally, you learned to style your functions so your programs will continue to be well-structured and as easy as possible for you and others to read.

Uno de tus objetivos como programador debe ser escribir código sencillo que haga lo que tú quieres, y las funciones te ayudan a conseguirlo. Te permiten escribir bloques de código y dejarlos en paz una vez que sabes que funcionan. Cuando sabes que una función hace su trabajo correctamente, puedes confiar en que seguirá funcionando y pasar a tu siguiente tarea de codificación.

One of your goals as a programmer should be to write simple code that does what you want it to, and functions help you do this. They allow you to write blocks of code and leave them alone once you know they work. When you know a function does its job correctly, you can trust that it will continue to work and move on to your next coding task.

Las funciones te permiten escribir código una vez y reutilizarlo tantas veces como quieras. Cuando necesites ejecutar el código de una función, todo lo que tienes que hacer es escribir una llamada de una línea y la función hará su trabajo. Cuando necesites modificar el comportamiento de una función, sólo tendrás que modificar un bloque de código, y tu cambio surtirá efecto en todos los lugares en los que hayas hecho una llamada a esa función.

Functions allow you to write code once and then reuse that code as many times as you want. When you need to run the code in a function, all you need to do is write a one-line call and the function does its job. When you need to modify a function's behavior, you only have to modify one block of code, and your change takes effect everywhere you've made a call to that function.

Utilizar funciones facilita la lectura de tus programas, y los buenos nombres de función resumen lo que hace cada parte de un programa. Leer una serie de llamadas a funciones te da una idea mucho más rápida de lo que hace un programa que leer una larga serie de bloques de código.

Using functions makes your programs easier to read, and good function names summarize what each part of a program does. Reading a series of function calls gives you a much quicker sense of what a program does than reading a long series of code blocks.

Las funciones también hacen que tu código sea más fácil de probar y depurar. Cuando la mayor parte del trabajo de tu programa lo realiza un conjunto de funciones, cada una de las cuales tiene un trabajo específico, es mucho más fácil probar y mantener el código que has escrito. Puedes escribir un programa independiente que llame a cada función y compruebe si cada función funciona en todas las situaciones que pueda encontrar. Al hacer esto, puedes estar seguro de que tus funciones funcionarán correctamente cada vez que las llares.

Functions also make your code easier to test and debug. When the bulk of your program's work is done by a set of functions, each of which has a specific job, it's much easier to test and maintain the code you've written. You can write a separate program that calls each function and tests whether each function works in all the situations it may encounter. When you do this, you can be confident that your functions will work properly each time you call them.

En el Capítulo 9, aprenderás a escribir clases. *Classes* combinan funciones y datos en un paquete ordenado que puede utilizarse de forma flexible y eficiente.

In Chapter 9, you'll learn to write classes. *Classes* combine functions and data into one neat package that can be used in flexible and efficient ways.

Capítulo 9: Clases (Chapter 9: Classes)

9

Clases (9 Classes)



DESCRIPCIÓN DE LA
IMAGEN (En español):

La imagen muestra una ilustración de una serpiente entrelazada alrededor de un motor a reacción. La serpiente tiene una expresión decidida. El motor se ve detallado con un diseño en blanco y negro.

Object-oriented programming (OOP) es uno de los enfoques más eficaces para escribir software. En la programación orientada a objetos, escribes que representan cosas y situaciones del mundo real, y creas basándote en estas clases. Cuando escribes una clase, defines el comportamiento general que puede tener toda una categoría de objetos. *classes* *objects*

Object-oriented programming (OOP) is one of the most effective approaches to writing software. In object-oriented programming, you write *classes* that represent real-world things and situations, and you create *objects* based on these classes. When you write a class, you define the general behavior that a whole category of objects can have.

Cuando creas objetos individuales a partir de la clase, cada objeto se equipa automáticamente con el comportamiento general; luego puedes dar a cada objeto los rasgos únicos que desees. Te sorprenderá lo bien que pueden modelarse situaciones del mundo real con la programación orientada a objetos.

When you create individual objects from the class, each object is automatically equipped with the general behavior; you can then give each object whatever unique traits you desire. You'll be amazed how well real-world situations can be modeled with object-oriented programming.

Crear un objeto a partir de una clase se llama *instantiation* , y trabajas con *instances* de una clase. En este capítulo escribirás clases y crearás instancias de esas clases. Especificarás el tipo de información que puede almacenarse en las instancias y definirás las acciones que pueden realizarse con ellas. También escribirás clases que amplíen la funcionalidad de clases existentes, de modo que clases similares puedan compartir una funcionalidad común, y puedas hacer más con menos código. Almacenarás tus clases en módulos e importarás clases escritas por otros programadores en tus propios archivos de programa.

Making an object from a class is called *instantiation* , and you work with *instances* of a class. In this chapter you'll write classes and create instances of those classes. You'll specify the kind of information that can be stored in instances, and you'll define actions that can be taken with these instances. You'll also write classes that extend the functionality of existing classes, so similar classes can share common functionality, and you can do more with less code. You'll store your classes in modules and import classes written by other programmers into your own program files.

Aprender sobre programación orientada a objetos te ayudará a ver el mundo como lo ve un programador. Te ayudará a entender tu código, no sólo lo que ocurre línea a línea, sino también los conceptos más amplios que hay detrás. Conocer la lógica que hay detrás de las clases te capacitará para pensar lógicamente, de modo que puedas escribir programas que aborden con eficacia casi cualquier problema que encuentres.

Learning about object-oriented programming will help you see the world as a programmer does. It'll help you understand your code—not just what's happening line by line, but also the bigger concepts behind it. Knowing the logic behind classes will train you to think logically, so you can write programs that effectively address almost any problem you encounter.

Las clases también te harán la vida más fácil a ti y a los demás programadores con los que trabajes cuando os enfrentéis a retos cada vez más complejos. Cuando tú y otros programadores escribáis código basado en el mismo tipo de lógica, podréis entender el trabajo de los demás. Tus programas tendrán sentido para las personas con las que trabajes, lo que permitirá a todos lograr más.

Classes also make life easier for you and the other programmers you'll work with as you take on increasingly complex challenges. When you and other programmers write code based on the same kind of logic, you'll be able to understand each other's work. Your

programs will make sense to the people you work with, allowing everyone to accomplish more.

Crear y utilizar una clase *(Creating and Using a Class)*

Puedes modelar casi cualquier cosa utilizando clases. Empecemos escribiendo una clase sencilla, `Dog`, que represente a un perro, no a un perro en particular, sino a cualquier perro. ¿Qué sabemos de la mayoría de los perros? Bueno, todos tienen un nombre y una edad. También sabemos que la mayoría de los perros se sientan y se dan la vuelta. Esos dos datos (nombre y edad) y esos dos comportamientos (sentarse y darse la vuelta) irán en nuestra clase `Dog` porque son comunes a la mayoría de los perros. Esta clase le dirá a Python cómo crear un objeto que represente a un perro. Una vez escrita nuestra clase, la utilizaremos para crear instancias individuales, cada una de las cuales representará a un perro concreto.

You can model almost anything using classes. Let's start by writing a simple class, `Dog`, that represents a dog—not one dog in particular, but any dog. What do we know about most pet dogs? Well, they all have a name and an age. We also know that most dogs sit and roll over. Those two pieces of information (name and age) and those two behaviors (sit and roll over) will go in our `Dog` class because they're common to most dogs. This class will tell Python how to make an object representing a dog. After our class is written, we'll use it to make individual instances, each of which represents one specific dog.

Creación de la clase Perro *(Creating the Dog Class)*

Cada instancia creada a partir de la clase `Dog` almacenará un `name` y un `age`, y daremos a cada perro la capacidad de `sit()` y `roll_over()`:

Each instance created from the `Dog` class will store a `name` and an `age`, and we'll give each dog the ability to `sit()` and `roll_over()`:

perro.py

dog.py

```

❶ class Dog:
    """A simple attempt to model a dog."""

    ❷ def __init__(self, name, age):
        """Initialize name and age attributes."""

    ❸ self.name = name
        self.age = age

    ❹ def sit(self):
        """Simulate a dog sitting in response to a command."""
        print(f'{self.name} is now sitting.')

    def roll_over(self):
        """Simulate rolling over in response to a command."""
        print(f'{self.name} rolled over!')

```

Hay mucho que observar aquí, pero no te preocupes. Verás esta estructura a lo largo de este capítulo y tendrás mucho tiempo para acostumbrarte a ella. Primero definimos una clase llamada `Dog` ❶. Por convención, los nombres en mayúsculas se refieren a clases en Python. No hay paréntesis en la definición de la clase porque la estamos creando desde cero. A continuación, escribimos un docstrip que describe lo que hace esta clase.

There's a lot to notice here, but don't worry. You'll see this structure throughout this chapter and have lots of time to get used to it. We first define a class called `Dog` ❶. By convention, capitalized names refer to classes in Python. There are no parentheses in the class definition because we're creating this class from scratch. We then write a docstring describing what this class does.

El método `__init__()` (*The `__init__()` Method*)

Una función que forma parte de una clase es un *method*. Todo lo que aprendiste sobre las funciones se aplica también a los métodos; la única diferencia práctica por ahora es la forma en que llamaremos a los métodos. El método `__init__()` ❷ es un método especial que Python ejecuta automáticamente cada vez que creamos una nueva instancia basada en la clase `Dog`. Este método tiene dos guiones bajos iniciales y dos guiones bajos finales, una convención que ayuda a evitar que los nombres de métodos por defecto de Python entren en conflicto con tus nombres de métodos. Asegúrate de utilizar dos guiones bajos a cada lado de `__init__()`. Si utilizas sólo uno a cada lado, el método no será llamado automáticamente cuando utilices tu clase, lo que puede dar lugar a errores difíciles de identificar.

A function that's part of a class is a *method* . Everything you learned about functions applies to methods as well; the only practical difference for now is the way we'll call methods. The `__init__()` method ❷ is a special method that Python runs automatically whenever we create a new instance based on the `Dog` class. This method has two leading underscores and two trailing underscores, a convention that helps prevent Python's default method names from conflicting with your method names. Make sure to use two underscores on each side of `__init__()` . If you use just one on each side, the method won't be called automatically when you use your class, which can result in errors that are difficult to identify.

Definimos el método `__init__()` para que tenga tres parámetros: `self` , `name` , y `age` . El parámetro `self` es obligatorio en la definición del método, y debe ir en primer lugar, antes que los demás parámetros. Debe incluirse en la definición porque cuando Python llame a este método más adelante (para crear una instancia de `Dog`), la llamada al método pasará automáticamente el argumento `self` . Cada llamada a un método asociado a una instancia pasa automáticamente `self` , que es una referencia a la propia instancia; da acceso a la instancia individual a los atributos y métodos de la clase. Cuando hagamos una instancia de `Dog` , Python llamará al método `__init__()` de la clase `Dog` . Pasaremos a `Dog()` un nombre y una edad como argumentos; `self` se pasa automáticamente, por lo que no necesitamos pasarlo. Siempre que queramos crear una instancia a partir de la clase `Dog` , proporcionaremos valores sólo para los dos últimos parámetros, `name` y `age` .

We define the `__init__()` method to have three parameters: `self` , `name` , and `age` . The `self` parameter is required in the method definition, and it must come first, before the other parameters. It must be included in the definition because when Python calls this method later (to create an instance of `Dog`), the method call will automatically pass the `self` argument. Every method call associated with an instance automatically passes `self` , which is a reference to the instance itself; it gives the individual instance access to the attributes and methods in the class. When we make an instance of `Dog` , Python will call the `__init__()` method from the `Dog` class. We'll pass `Dog()` a name and an age as arguments; `self` is passed automatically, so we don't need to pass it. Whenever we want to make an instance from the `Dog` class, we'll provide values for only the last two parameters, `name` and `age` .

Las dos variables definidas en el cuerpo del método `__init__()` tienen el prefijo `self` ❸ . Cualquier variable prefijada con `self` está disponible para todos los métodos de la clase, y también podremos acceder a estas variables a través de cualquier instancia creada a partir de la clase. La línea `self.name = name` toma el valor asociado al parámetro `name` y lo asigna a la variable `name` , que se adjunta a la instancia que se está creando. El mismo

proceso ocurre con `self.age = age` . Las variables accesibles a través de instancias como ésta se denominan *attributes* .

The two variables defined in the body of the `__init__()` method each have the prefix `self` ③ . Any variable prefixed with `self` is available to every method in the class, and we'll also be able to access these variables through any instance created from the class. The line `self.name = name` takes the value associated with the parameter `name` and assigns it to the variable `name` , which is then attached to the instance being created. The same process happens with `self.age = age` . Variables that are accessible through instances like this are called *attributes* .

La clase `Dog` tiene definidos otros dos métodos: `sit()` y `roll_over()` ④ . Como estos métodos no necesitan información adicional para ejecutarse, sólo los definimos para que tengan un parámetro, `self` . Las instancias que creemos más adelante tendrán acceso a estos métodos. En otras palabras, podrán sentarse y darse la vuelta. Por ahora, `sit()` y `roll_over()` no hacen gran cosa. Simplemente imprimen un mensaje diciendo que el perro se sienta o se da la vuelta. Pero el concepto puede ampliarse a situaciones realistas: si esta clase formara parte de un juego de ordenador, estos métodos contendrían código para hacer que un perro animado se sentara y se diera la vuelta. Si esta clase se escribiera para controlar un robot, estos métodos dirigirían los movimientos que hacen que un perro robótico se siente y se dé la vuelta.

The `Dog` class has two other methods defined: `sit()` and `roll_over()` ④ . Because these methods don't need additional information to run, we just define them to have one parameter, `self` . The instances we create later will have access to these methods. In other words, they'll be able to sit and roll over. For now, `sit()` and `roll_over()` don't do much. They simply print a message saying the dog is sitting or rolling over. But the concept can be extended to realistic situations: if this class were part of a computer game, these methods would contain code to make an animated dog sit and roll over. If this class was written to control a robot, these methods would direct movements that cause a robotic dog to sit and roll over.

Crear una instancia a partir de una clase (*Making an Instance from a Class*)

Piensa en una clase como un conjunto de instrucciones sobre cómo crear una instancia. La clase `Dog` es un conjunto de instrucciones que indica a Python cómo crear instancias individuales que representen a perros concretos.

Think of a class as a set of instructions for how to make an instance. The `Dog` class is a set of instructions that tells Python how to make individual instances representing specific dogs.

Hagamos una instancia que represente a un perro concreto:

Let's make an instance representing a specific dog:

```
class Dog:  
    --snip--
```

```
❶ my_dog = Dog('Willie', 6)  
  
❷ print(f'My dog's name is {my_dog.name}.')  
❸ print(f'My dog is {my_dog.age} years old.')
```

La clase `Dog` que utilizamos aquí es la que acabamos de escribir en el ejemplo anterior. Aquí, le decimos a Python que cree un perro cuyo nombre sea `'Willie'` y cuya edad sea `6` **❶**. Cuando Python lee esta línea, llama al método `__init__()` de `Dog` con los argumentos `'Willie'` y `6`. El método `__init__()` crea una instancia que representa a este perro concreto y establece los atributos `name` y `age` utilizando los valores que le hemos proporcionado. Python devuelve entonces una instancia que representa a este perro. Asignamos esa instancia a la variable `my_dog`. La convención de nomenclatura es útil aquí; normalmente podemos asumir que un nombre en mayúsculas como `Dog` se refiere a una clase, y un nombre en minúsculas como `my_dog` se refiere a una única instancia creada a partir de una clase.

The `Dog` class we're using here is the one we just wrote in the previous example. Here, we tell Python to create a dog whose name is `'Willie'` and whose age is `6` **❶**. When Python reads this line, it calls the `__init__()` method in `Dog` with the arguments `'Willie'` and `6`. The `__init__()` method creates an instance representing this particular dog and sets the `name` and `age` attributes using the values we provided. Python then returns an instance representing this dog. We assign that instance to the variable `my_dog`. The naming convention is helpful here; we can usually assume that a capitalized name like `Dog` refers to a class, and a lowercase name like `my_dog` refers to a single instance created from a class.

Acceder a los atributos (Accessing Attributes)

Para acceder a los atributos de una instancia, se utiliza la notación con puntos. Accedemos al valor del atributo `my_dog` 's `name` ❷ escribiendo:

To access the attributes of an instance, you use dot notation. We access the value of `my_dog` 's attribute `name` ❷ by writing:

```
my_dog.name
```

La notación con puntos se utiliza a menudo en Python. Esta sintaxis muestra cómo Python encuentra el valor de un atributo. Aquí, Python busca en la instancia `my_dog` y luego encuentra el atributo `name` asociado a `my_dog`. Se trata del mismo atributo denominado `self.name` en la clase `Dog`. Utilizamos el mismo enfoque para trabajar con el atributo `age` ❸.

Dot notation is used often in Python. This syntax demonstrates how Python finds an attribute's value. Here, Python looks at the instance `my_dog` and then finds the attribute `name` associated with `my_dog`. This is the same attribute referred to as `self.name` in the class `Dog`. We use the same approach to work with the attribute `age` ❸.

El resultado es un resumen de lo que sabemos sobre `my_dog` :

The output is a summary of what we know about `my_dog` :

```
My dog's name is Willie.  
My dog is 6 years old.
```

Llamada a métodos (Calling Methods)

Después de crear una instancia de la clase `Dog`, podemos utilizar la notación punto para llamar a cualquier método definido en `Dog`. Hagamos que nuestro perro se siente y se dé la vuelta:

After we create an instance from the class `Dog` , we can use dot notation to call any method defined in `Dog` . Let's make our dog sit and roll over:

```
class Dog:
    --snip--

my_dog = Dog('Willie', 6)
my_dog.sit()
my_dog.roll_over()
```

Para llamar a un método, indica el nombre de la instancia (en este caso, `my_dog`) y el método al que quieres llamar, separados por un punto. Cuando Python lee `my_dog.sit()` , busca el método `sit()` en la clase `Dog` y ejecuta ese código. Python interpreta la línea `my_dog.roll_over()` del mismo modo.

To call a method, give the name of the instance (in this case, `my_dog`) and the method you want to call, separated by a dot. When Python reads `my_dog.sit()` , it looks for the method `sit()` in the class `Dog` and runs that code. Python interprets the line `my_dog.roll_over()` in the same way.

Ahora Willie hace lo que le decimos:

Now Willie does what we tell him to:

```
Willie is now sitting.
Willie rolled over!
```

Esta sintaxis es bastante útil. Cuando a los atributos y métodos se les han dado nombres adecuadamente descriptivos como `name` , `age` , `sit()` , y `roll_over()` , podemos deducir fácilmente lo que se supone que debe hacer un bloque de código, incluso uno que nunca hemos visto antes.

This syntax is quite useful. When attributes and methods have been given appropriately descriptive names like `name` , `age` , `sit()` , and `roll_over()` , we can easily infer what a block of code, even one we've never seen before, is supposed to do.

Crear varias instancias (Creating Multiple Instances)

Puedes crear tantas instancias de una clase como necesites. Vamos a crear un segundo perro llamado `your_dog` :

You can create as many instances from a class as you need. Let's create a second dog called `your_dog` :

```
class Dog:
    --snip--

my_dog = Dog('Willie', 6)
your_dog = Dog('Lucy', 3)

print(f'My dog's name is {my_dog.name}.')
print(f'My dog is {my_dog.age} years old.')
my_dog.sit()

print(f'\nYour dog's name is {your_dog.name}.')
print(f'Your dog is {your_dog.age} years old.')
your_dog.sit()
```

En este ejemplo creamos un perro llamado Willie y un perro llamado Lucy. Cada perro es una instancia independiente con su propio conjunto de atributos, capaz del mismo conjunto de acciones:

In this example we create a dog named Willie and a dog named Lucy. Each dog is a separate instance with its own set of attributes, capable of the same set of actions:

My dog's name is Willie.
My dog is 6 years old.
Willie is now sitting.

Your dog's name is Lucy.
Your dog is 3 years old.
Lucy is now sitting.

Aunque utilizáramos el mismo nombre y la misma edad para el segundo perro, Python seguiría creando una instancia independiente a partir de la clase `Dog`. Puedes crear tantas instancias de una clase como necesites, siempre que des a cada instancia un nombre de variable único o que ocupe un lugar único en una lista o diccionario.

Even if we used the same name and age for the second dog, Python would still create a separate instance from the `Dog` class. You can make as many instances from one class as you need, as long as you give each instance a unique variable name or it occupies a unique spot in a list or dictionary.

Pruébalo tú mismo (*Try It Yourself*)

9-1. Restaurante: Crea una clase llamada `Restaurant`. El método `__init__()` de `Restaurant` debe almacenar dos atributos: un `restaurant_name` y un `cuisine_type`. Crea un método llamado `describe_restaurant()` que imprima estos dos datos, y un método llamado `open_restaurant()` que imprima un mensaje indicando que el restaurante está abierto.

9-1. Restaurant: Make a class called `Restaurant`. The `__init__()` method for `Restaurant` should store two attributes: a `restaurant_name` and a `cuisine_type`. Make a method called `describe_restaurant()` that prints these two pieces of information, and a method called `open_restaurant()` that prints a message indicating that the restaurant is open.

Crea una instancia llamada `restaurant` a partir de tu clase. Imprime los dos atributos individualmente, y luego llama a ambos métodos.

Make an instance called `restaurant` from your class. Print the two attributes individually, and then call both methods.

9-2. Tres Restaurantes: Empieza con tu clase del Ejercicio 9-1. Crea tres instancias diferentes a partir de la clase, y llama a `describe_restaurant()` para cada instancia.

9-2. Three Restaurants: Start with your class from Exercise 9-1. Create three different instances from the class, and call `describe_restaurant()` for each instance.

9-3. Usuarios: Crea una clase llamada `User` . Crea dos atributos llamados `first_name` y `last_name` , y luego crea otros atributos que se suelen almacenar en un perfil de usuario. Crea un método llamado `describe_user()` que imprima un resumen de la información del usuario. Crea otro método llamado `greet_user()` que imprima un saludo personalizado al usuario.

9-3. Users: Make a class called `User` . Create two attributes called `first_name` and `last_name` , and then create several other attributes that are typically stored in a user profile. Make a method called `describe_user()` that prints a summary of the user's information. Make another method called `greet_user()` that prints a personalized greeting to the user.

Crea varias instancias que representen a distintos usuarios, y llama a ambos métodos para cada usuario.

Create several instances representing different users, and call both methods for each user.

Trabajar con clases e instancias *(Working with Classes and Instances)*

Puedes utilizar clases para representar muchas situaciones del mundo real. Una vez que escribas una clase, pasarás la mayor parte del tiempo trabajando con instancias creadas a partir de esa clase. Una de las primeras tareas que querrás hacer es modificar los atributos asociados a una instancia concreta. Puedes modificar los atributos de una instancia directamente o escribir métodos que actualicen los atributos de formas específicas.

You can use classes to represent many real-world situations. Once you write a class, you'll spend most of your time working with instances created from that class. One of the first tasks you'll want to do is modify the attributes associated with a particular instance. You can modify the attributes of an instance directly or write methods that update attributes in specific ways.

La clase Coche (*The Car Class*)

Escribamos una nueva clase que represente a un coche. Nuestra clase almacenará información sobre el tipo de coche con el que estamos trabajando, y tendrá un método que resuma esta información:

Let's write a new class representing a car. Our class will store information about the kind of car we're working with, and it will have a method that summarizes this information:

coche.py

car.py

```
class Car:
    """A simple attempt to represent a car."""

    ❶ def __init__(self, make, model, year):
        """Initialize attributes to describe a car."""
        self.make = make
        self.model = model
        self.year = year

    ❷ def get_descriptive_name(self):
        """Return a neatly formatted descriptive name."""
        long_name = f'{self.year} {self.make} {self.model}'
        return long_name.title()

    ❸ my_new_car = Car('audi', 'a4', 2024)
    print(my_new_car.get_descriptive_name())
```

En la clase `Car`, definimos el método `__init__()` con el parámetro `self` primero ❶, igual que hicimos con la clase `Dog`. También le damos otros tres parámetros: `make`, `model`, y `year`. El método `__init__()` toma estos parámetros y los asigna a los atributos que se asociarán a las instancias creadas a partir de esta clase. Cuando creamos una nueva instancia de `Car`, tendremos que especificar una marca, un modelo y un año para nuestra instancia.

In the `Car` class, we define the `__init__()` method with the `self` parameter first ❶, just like we did with the `Dog` class. We also give it three other parameters: `make`, `model`, and `year`. The `__init__()` method takes in these parameters and assigns them to the attributes that will be associated with instances made from this class. When we make a new `Car` instance, we'll need to specify a `make`, `model`, and `year` for our instance.

Definimos un método llamado `get_descriptive_name()` ❷ que pone los datos `year`, `make` y `model` de un coche en una sola cadena que describe claramente el coche. Esto nos evitará tener que imprimir individualmente el valor de cada atributo. Para trabajar con los valores de los atributos en este método, utilizamos `self.make`, `self.model`, y `self.year`. Fuera de la clase, creamos una instancia de la clase `Car` y la asignamos a la variable `my_new_car` ❸. Luego llamamos a `get_descriptive_name()` para mostrar qué clase de coche tenemos:

We define a method called `get_descriptive_name()` ❷ that puts a car's `year`, `make`, and `model` into one string neatly describing the car. This will spare us from having to print each attribute's value individually. To work with the attribute values in this method, we use `self.make`, `self.model`, and `self.year`. Outside of the class, we make an instance from the `Car` class and assign it to the variable `my_new_car` ❸. Then we call `get_descriptive_name()` to show what kind of car we have:

2024 Audi A4

Para hacer la clase más interesante, añadamos un atributo que cambie con el tiempo. Añadiremos un atributo que almacene el kilometraje total del coche.

To make the class more interesting, let's add an attribute that changes over time. We'll add an attribute that stores the car's overall mileage.

Establecer un valor por defecto para un atributo (*Setting a Default Value for an Attribute*)

Cuando se crea una instancia, se pueden definir atributos sin pasarlos como parámetros. Estos atributos pueden definirse en el método `__init__()`, donde se les asigna un valor por defecto.

When an instance is created, attributes can be defined without being passed in as parameters. These attributes can be defined in the `__init__()` method, where they are assigned a default value.

Vamos a añadir un atributo llamado `odometer_reading` que comience siempre con el valor 0. También añadiremos un método `read_odometer()` que nos ayude a leer el cuentakilómetros de cada coche:

Let's add an attribute called `odometer_reading` that always starts with a value of 0. We'll also add a method `read_odometer()` that helps us read each car's odometer:

class Car:

```
def __init__(self, make, model, year):
```

```
    """Initialize attributes to describe a car."""
```

```
    self.make = make
```

```
    self.model = model
```

```
    self.year = year
```

```
❶ self.odometer_reading = 0
```

```
def get_descriptive_name(self):
```

```
    --snip--
```

```
❷ def read_odometer(self):
```

```
    """Print a statement showing the car's mileage."""
```

```
    print(f"This car has {self.odometer_reading} miles on it.")
```

```
my_new_car = Car('audi', 'a4', 2024)
```

```
print(my_new_car.get_descriptive_name())
```

```
my_new_car.read_odometer()
```

Esta vez, cuando Python llama al método `__init__()` para crear una nueva instancia, almacena los valores de marca, modelo y año como atributos, como hizo en el ejemplo anterior. A continuación, Python crea un nuevo atributo llamado `odometer_reading` y establece su valor inicial en 0 ❶. También tenemos un nuevo método llamado `read_odometer()` ❷ que facilita la lectura del kilometraje de un coche.

This time, when Python calls the `__init__()` method to create a new instance, it stores the make, model, and year values as attributes, like it did in the previous example. Then Python creates a new attribute called `odometer_reading` and sets its initial value to 0 ❶. We also have a new method called `read_odometer()` ❷ that makes it easy to read a car's mileage.

Nuestro coche empieza con un kilometraje de 0:

Our car starts with a mileage of 0:

2024 Audi A4

This car has 0 miles on it.

No se venden muchos coches con exactamente 0 kilómetros en el cuentakilómetros, así que necesitamos una forma de cambiar el valor de este atributo.

Not many cars are sold with exactly 0 miles on the odometer, so we need a way to change the value of this attribute.

Modificar los valores de los atributos *(Modifying Attribute Values)*

Puedes cambiar el valor de un atributo de tres formas: puedes cambiar el valor directamente a través de una instancia, establecer el valor a través de un método o incrementar el valor (añadirle una determinada cantidad) a través de un método. Veamos cada uno de estos enfoques.

You can change an attribute's value in three ways: you can change the value directly through an instance, set the value through a method, or increment the value (add a certain amount to it) through a method. Let's look at each of these approaches.

Modificar directamente el valor de un atributo *(Modifying an Attribute's Value Directly)*

La forma más sencilla de modificar el valor de un atributo es acceder a él directamente a través de una instancia. Aquí fijamos directamente la lectura del cuentakilómetros en 23:

The simplest way to modify the value of an attribute is to access the attribute directly through an instance. Here we set the odometer reading to 23 directly:

```
class Car:
    --snip--

my_new_car = Car('audi', 'a4', 2024)
print(my_new_car.get_descriptive_name())

my_new_car.odometer_reading = 23
my_new_car.read_odometer()
```

Utilizamos la notación con puntos para acceder al atributo `odometer_reading` del coche, y establecemos su valor directamente. Esta línea le dice a Python que tome la instancia `my_new_car`, busque el atributo `odometer_reading` asociado a ella y establezca el valor de ese atributo en 23:

We use dot notation to access the car's `odometer_reading` attribute, and set its value directly. This line tells Python to take the instance `my_new_car`, find the attribute `odometer_reading` associated with it, and set the value of that attribute to 23:

2024 Audi A4

This car has 23 miles on it.

A veces querrás acceder directamente a los atributos de esta forma, pero otras veces querrás escribir un método que actualice el valor por ti.

Sometimes you'll want to access attributes directly like this, but other times you'll want to write a method that updates the value for you.

[Modificar el valor de un atributo mediante un método \(Modifying an Attribute's Value Through a Method\)](#)

Puede ser útil disponer de métodos que actualicen determinados atributos por ti. En lugar de acceder directamente al atributo, pasas el nuevo valor a un método que se encarga internamente de la actualización.

It can be helpful to have methods that update certain attributes for you. Instead of accessing the attribute directly, you pass the new value to a method that handles the updating internally.

Aquí tienes un ejemplo de un método llamado `update_odometer()` :

Here's an example showing a method called `update_odometer()` :

```
class Car:
    --snip--

    def update_odometer(self, mileage):
        """Set the odometer reading to the given value."""
        self.odometer_reading = mileage

my_new_car = Car('audi', 'a4', 2024)
print(my_new_car.get_descriptive_name())

❶ my_new_car.update_odometer(23)
my_new_car.read_odometer()
```

La única modificación de `Car` es la adición de `update_odometer()` . Este método toma un valor de kilometraje y lo asigna a `self.odometer_reading` . Utilizando la instancia `my_new_car` , llamamos a `update_odometer()` con 23 como argumento ❶ . Esto establece la lectura del cuentakilómetros en 23, y `read_odometer()` imprime la lectura:

The only modification to `Car` is the addition of `update_odometer()` . This method takes in a mileage value and assigns it to `self.odometer_reading` . Using the `my_new_car` instance, we call `update_odometer()` with 23 as an argument ❶ . This sets the odometer reading to 23, and `read_odometer()` prints the reading:

2024 Audi A4

This car has 23 miles on it.

Podemos ampliar el método `update_odometer()` para que realice un trabajo adicional cada vez que se modifique la lectura del cuentakilómetros. Añadamos un poco de lógica para asegurarnos de que nadie intenta hacer retroceder la lectura del cuentakilómetros:

We can extend the method `update_odometer()` to do additional work every time the odometer reading is modified. Let's add a little logic to make sure no one tries to roll back the odometer reading:

```
class Car:
    --snip--

    def update_odometer(self, mileage):
        """
        Set the odometer reading to the given value.
        Reject the change if it attempts to roll the odometer back.
        """
        ❶ if mileage >= self.odometer_reading:
            self.odometer_reading = mileage
        else:
            ❷ print("You can't roll back an odometer!")
```

Ahora `update_odometer()` comprueba que la nueva lectura tiene sentido antes de modificar el atributo. Si el valor proporcionado para `mileage` es mayor o igual que el kilometraje existente, `self.odometer_reading`, puedes actualizar la lectura del cuentakilómetros al nuevo kilometraje ❶. Si el nuevo kilometraje es menor que el kilometraje existente, recibirás un aviso de que no puedes retroceder un cuentakilómetros ❷.

Now `update_odometer()` checks that the new reading makes sense before modifying the attribute. If the value provided for `mileage` is greater than or equal to the existing `mileage`, `self.odometer_reading`, you can update the odometer reading to the new mileage ❶. If the new mileage is less than the existing mileage, you'll get a warning that you can't roll back an odometer ❷.

Incrementar el valor de un atributo mediante un método (Incrementing an Attribute's Value Through a Method)

A veces querrás incrementar el valor de un atributo en una cantidad determinada, en lugar de establecer un valor completamente nuevo. Supongamos que compramos un coche usado y le ponemos 160 km entre el momento en que lo compramos y el momento en que lo matriculamos. Aquí tenemos un método que nos permite pasar esta cantidad incremental y añadir ese valor a la lectura del cuentakilómetros:

Sometimes you'll want to increment an attribute's value by a certain amount, rather than set an entirely new value. Say we buy a used car and put 100 miles on it between the time we buy it and the time we register it. Here's a method that allows us to pass this incremental amount and add that value to the odometer reading:

```
class Car:
    --snip--

    def update_odometer(self, mileage):
        --snip--

    def increment_odometer(self, miles):
        """Add the given amount to the odometer reading."""
        self.odometer_reading += miles

❶ my_used_car = Car('subaru', 'outback', 2019)
print(my_used_car.get_descriptive_name())

❷ my_used_car.update_odometer(23_500)
my_used_car.read_odometer()

my_used_car.increment_odometer(100)
my_used_car.read_odometer()
```

El nuevo método `increment_odometer()` toma un número de kilómetros, y añade este valor a `self.odometer_reading`. Primero, creamos un coche usado, `my_used_car` ❶. Ponemos su cuentakilómetros en 23.500 llamando a `update_odometer()` y pasándole 23_500 ❷. Por último, llamamos a `increment_odometer()` y le pasamos 100 para añadir los 160 km que recorrimos entre la compra del coche y su matriculación:

The new method `increment_odometer()` takes in a number of miles, and adds this value to `self.odometer_reading`. First, we create a used car, `my_used_car` ❶. We set its odometer to 23,500 by calling `update_odometer()` and passing it 23_500 ❷. Finally, we call `increment_odometer()` and pass it 100 to add the 100 miles that we drove between buying the car and registering it:

2019 Subaru Outback

This car has 23500 miles on it.

This car has 23600 miles on it.

Puedes modificar este método para que rechace los incrementos negativos, de modo que nadie utilice también esta función para hacer retroceder un cuentakilómetros.

You can modify this method to reject negative increments so no one uses this function to roll back an odometer as well.

Nota (Note)

Puedes utilizar métodos como éste para controlar cómo los usuarios de tu programa actualizan valores como la lectura de un cuentakilómetros, pero cualquiera con acceso al programa puede establecer la lectura del cuentakilómetros en cualquier valor accediendo directamente al atributo. Una seguridad eficaz requiere una atención extrema a los detalles, además de comprobaciones básicas como las que se muestran aquí.

You can use methods like this to control how users of your program update values such as an odometer reading, but anyone with access to the program can set the odometer reading to any value by accessing the attribute directly. Effective security takes extreme attention to detail in addition to basic checks like those shown here.

Pruébalo tú mismo (Try It Yourself)

9-4. Número servido: Empieza con tu programa del Ejercicio 9-1 (página 162). Añade un atributo llamado `number_served` con un valor por defecto de 0. Crea una instancia llamada `restaurant` a partir de esta clase. Imprime el número de clientes a los que ha servido el restaurante, y luego cambia este valor e imprímelo de nuevo.

9-4. Number Served: Start with your program from Exercise 9-1 (page 162). Add an attribute called `number_served` with a default value of 0. Create an instance called

restaurant from this class. Print the number of customers the restaurant has served, and then change this value and print it again.

Añade un método llamado `set_number_served()` que te permita establecer el número de clientes a los que se ha servido. Llama a este método con un nuevo número e imprime de nuevo el valor.

Add a method called `set_number_served()` that lets you set the number of customers that have been served. Call this method with a new number and print the value again.

Añade un método llamado `increment_number_served()` que te permita incrementar el número de clientes que han sido atendidos. Llama a este método con cualquier número que quieras que pueda representar cuántos clientes han sido atendidos en, digamos, un día de negocio.

Add a method called `increment_number_served()` that lets you increment the number of customers who've been served. Call this method with any number you like that could represent how many customers were served in, say, a day of business.

9-5. Intentos de inicio de sesión: Añade un atributo llamado `login_attempts` a tu clase `User` del Ejercicio 9-3 (página 162). Escribe un método llamado `increment_login_attempts()` que incremente el valor de `login_attempts` en 1. Escribe otro método llamado `reset_login_attempts()` que restablezca el valor de `login_attempts` a 0.

9-5. Login Attempts: Add an attribute called `login_attempts` to your `User` class from Exercise 9-3 (page 162). Write a method called `increment_login_attempts()` that increments the value of `login_attempts` by 1. Write another method called `reset_login_attempts()` that resets the value of `login_attempts` to 0.

Crea una instancia de la clase `User` y llama varias veces a `increment_login_attempts()`. Imprime el valor de `login_attempts` para asegurarte de que se ha incrementado correctamente, y luego llama a `reset_login_attempts()`. Vuelve a imprimir `login_attempts` para asegurarte de que se ha puesto a 0.

Make an instance of the `User` class and call `increment_login_attempts()` several times. Print the value of `login_attempts` to make sure it was incremented properly, and then call `reset_login_attempts()`. Print `login_attempts` again to make sure it was reset to 0.

Herencia (*Inheritance*)

No siempre tienes que empezar de cero al escribir una clase. Si la clase que estás escribiendo es una versión especializada de otra clase que escribiste, puedes utilizar *inheritance*. Cuando una clase *inherits* de otra, adopta los atributos y métodos de la primera clase. La clase original se llama *parent class*, y la nueva clase es *child class*. La clase hija puede heredar alguno o todos los atributos y métodos de su clase padre, pero también es libre de definir nuevos atributos y métodos propios.

You don't always have to start from scratch when writing a class. If the class you're writing is a specialized version of another class you wrote, you can use *inheritance*. When one class *inherits* from another, it takes on the attributes and methods of the first class. The original class is called the *parent class*, and the new class is the *child class*. The child class can inherit any or all of the attributes and methods of its parent class, but it's also free to define new attributes and methods of its own.

El método `__init__()` de una clase hija (*The `__init__()` Method for a Child Class*)

Cuando escribas una nueva clase basada en una clase existente, a menudo querrás llamar al método `__init__()` de la clase padre. Esto inicializará los atributos definidos en el método `__init__()` de la clase padre y los pondrá a disposición de la clase hija.

When you're writing a new class based on an existing class, you'll often want to call the `__init__()` method from the parent class. This will initialize any attributes that were defined in the parent `__init__()` method and make them available in the child class.

Por ejemplo, modelemos un coche eléctrico. Un coche eléctrico es sólo una clase específica de coche, así que podemos basar nuestra nueva clase `ElectricCar` en la clase `Car` que escribimos antes. Así sólo tendremos que escribir código para los atributos y comportamientos específicos de los coches eléctricos.

As an example, let's model an electric car. An electric car is just a specific kind of car, so we can base our new `ElectricCar` class on the `Car` class we wrote earlier. Then we'll only have to write code for the attributes and behaviors specific to electric cars.

Empecemos haciendo una versión sencilla de la clase `ElectricCar`, que hace todo lo que hace la clase `Car`:

Let's start by making a simple version of the `ElectricCar` class, which does everything the `Car` class does:

coche_eléctrico.py

electric_car.py

```
❶ class Car:
    """A simple attempt to represent a car."""

    def __init__(self, make, model, year):
        """Initialize attributes to describe a car."""
        self.make = make
        self.model = model
        self.year = year
        self.odometer_reading = 0

    def get_descriptive_name(self):
        """Return a neatly formatted descriptive name."""
        long_name = f"{self.year} {self.make} {self.model}"
        return long_name.title()

    def read_odometer(self):
        """Print a statement showing the car's mileage."""
        print(f"This car has {self.odometer_reading} miles on it.")

    def update_odometer(self, mileage):
```

```

        """Set the odometer reading to the given value."""
        if mileage >= self.odometer_reading:
            self.odometer_reading = mileage
        else:
            print("You can't roll back an odometer!")

    def increment_odometer(self, miles):
        """Add the given amount to the odometer reading."""
        self.odometer_reading += miles

❷ class ElectricCar(Car):
    """Represent aspects of a car, specific to electric vehicles."""

    ❸ def __init__(self, make, model, year):
        """Initialize attributes of the parent class."""
    ❹     super().__init__(make, model, year)

❺ my_leaf = ElectricCar('nissan', 'leaf', 2024)
print(my_leaf.get_descriptive_name())

```

Empezamos con `Car` ❶ . Cuando creas una clase hija, la clase padre debe formar parte del archivo actual y debe aparecer antes que la clase hija en el archivo. A continuación, definimos la clase hija, `ElectricCar` ❷ . El nombre de la clase padre debe incluirse entre paréntesis en la definición de una clase hija. El método `__init__()` recoge la información necesaria para crear una instancia de `Car` ❸ .

We start with `Car` ❶ . When you create a child class, the parent class must be part of the current file and must appear before the child class in the file. We then define the child class, `ElectricCar` ❷ . The name of the parent class must be included in parentheses in the definition of a child class. The `__init__()` method takes in the information required to make a `Car` instance ❸ .

La función `super()` ❹ es una función especial que te permite llamar a un método de la clase padre. Esta línea indica a Python que llame al método `__init__()` desde `Car` , lo que proporciona a una instancia `ElectricCar` todos los atributos definidos en ese método. El nombre *super* proviene de la convención de llamar a la clase padre *superclass* y a la clase hija *subclass* .

The `super()` function ④ is a special function that allows you to call a method from the parent class. This line tells Python to call the `__init__()` method from `Car`, which gives an `ElectricCar` instance all the attributes defined in that method. The name *super* comes from a convention of calling the parent class a *superclass* and the child class a *subclass*.

Comprobamos si la herencia funciona correctamente intentando crear un coche eléctrico con el mismo tipo de información que proporcionaríamos al crear un coche normal. Creamos una instancia de la clase `ElectricCar` y la asignamos a `my_leaf` ⑤. Esta línea llama al método `__init__()` definido en `ElectricCar`, que a su vez le dice a Python que llame al método `__init__()` definido en la clase padre `Car`. Proporcionamos los argumentos `'nissan'`, `'leaf'`, y `2024`.

We test whether inheritance is working properly by trying to create an electric car with the same kind of information we'd provide when making a regular car. We make an instance of the `ElectricCar` class and assign it to `my_leaf` ⑤. This line calls the `__init__()` method defined in `ElectricCar`, which in turn tells Python to call the `__init__()` method defined in the parent class `Car`. We provide the arguments `'nissan'`, `'leaf'`, and `2024`.

Aparte de `__init__()`, todavía no hay atributos ni métodos particulares de un coche eléctrico. En este momento sólo nos estamos asegurando de que el coche eléctrico tiene los comportamientos `Car` adecuados:

Aside from `__init__()`, there are no attributes or methods yet that are particular to an electric car. At this point we're just making sure the electric car has the appropriate `Car` behaviors:

2024 Nissan Leaf

La instancia de `ElectricCar` funciona igual que una instancia de `Car`, así que ahora podemos empezar a definir atributos y métodos específicos de los coches eléctricos.

The `ElectricCar` instance works just like an instance of `Car`, so now we can begin defining attributes and methods specific to electric cars.

Definir atributos y métodos para la clase hija (*Defining Attributes and Methods for the Child Class*)

Una vez que tengas una clase hija que hereda de una clase padre, puedes añadir los atributos y métodos nuevos que sean necesarios para diferenciar la clase hija de la clase padre.

Once you have a child class that inherits from a parent class, you can add any new attributes and methods necessary to differentiate the child class from the parent class.

Añadamos un atributo específico de los coches eléctricos (una batería, por ejemplo) y un método para informar sobre este atributo. Almacenaremos el tamaño de la batería y escribiremos un método que imprima una descripción de la batería:

Let's add an attribute that's specific to electric cars (a battery, for example) and a method to report on this attribute. We'll store the battery size and write a method that prints a description of the battery:

```
class Car:
    --snip--

class ElectricCar(Car):
    """Represent aspects of a car, specific to electric vehicles."""

    def __init__(self, make, model, year):
        """
        Initialize attributes of the parent class.
        Then initialize attributes specific to an electric car.
        """
        super().__init__(make, model, year)
        ❶ self.battery_size = 40

        ❷ def describe_battery(self):
            """Print a statement describing the battery size."""
            print(f"This car has a {self.battery_size}-kWh battery.")

my_leaf = ElectricCar('nissan', 'leaf', 2024)
print(my_leaf.get_descriptive_name())
my_leaf.describe_battery()
```

Añadimos un nuevo atributo `self.battery_size` y establecemos su valor inicial en 40 ❶ . Este atributo se asociará a todas las instancias creadas a partir de la clase `ElectricCar` pero no se asociará a ninguna instancia de `Car` . También añadimos un método llamado `describe_battery()` que imprime información sobre la pila ❷ . Cuando llamamos a este método, obtenemos una descripción claramente específica de un coche eléctrico:

We add a new attribute `self.battery_size` and set its initial value to 40 ❶ . This attribute will be associated with all instances created from the `ElectricCar` class but won't be associated with any instances of `Car` . We also add a method called `describe_battery()` that prints information about the battery ❷ . When we call this method, we get a description that is clearly specific to an electric car:

2024 Nissan Leaf

This car has a 40-kWh battery.

No hay límite en cuanto a lo que puedes especializar la clase `ElectricCar` . Puedes añadir tantos atributos y métodos como necesites para modelar un coche eléctrico con el grado de precisión que necesites. Un atributo o método que podría pertenecer a cualquier coche, en lugar de uno específico de un coche eléctrico, debería añadirse a la clase `Car` en lugar de a la clase `ElectricCar` . Así, cualquiera que utilice la clase `Car` tendrá también disponible esa funcionalidad, y la clase `ElectricCar` sólo contendrá código para la información y el comportamiento específicos de los vehículos eléctricos.

There's no limit to how much you can specialize the `ElectricCar` class. You can add as many attributes and methods as you need to model an electric car to whatever degree of accuracy you need. An attribute or method that could belong to any car, rather than one that's specific to an electric car, should be added to the `Car` class instead of the `ElectricCar` class. Then anyone who uses the `Car` class will have that functionality available as well, and the `ElectricCar` class will only contain code for the information and behavior specific to electric vehicles.

Reemplazar métodos de la clase padre (*Overriding Methods from the Parent Class*)

Puedes anular cualquier método de la clase padre que no se ajuste a lo que intentas modelar con la clase hija. Para ello, define un método en la clase hija con el mismo nombre que el método que quieres anular en la clase padre. Python ignorará el método de la clase padre y sólo prestará atención al método que definas en la clase hija.

You can override any method from the parent class that doesn't fit what you're trying to model with the child class. To do this, you define a method in the child class with the same name as the method you want to override in the parent class. Python will disregard the parent class method and only pay attention to the method you define in the child class.

Supongamos que la clase `Car` tiene un método llamado `fill_gas_tank()`. Este método no tiene sentido para un vehículo totalmente eléctrico, así que tal vez quieras anular este método. He aquí una forma de hacerlo:

Say the class `Car` had a method called `fill_gas_tank()`. This method is meaningless for an all-electric vehicle, so you might want to override this method. Here's one way to do that:

```
class ElectricCar(Car):
    --snip--

    def fill_gas_tank(self):
        """Electric cars don't have gas tanks."""
        print("This car doesn't have a gas tank!")
```

Ahora, si alguien intenta llamar a `fill_gas_tank()` con un coche eléctrico, Python ignorará el método `fill_gas_tank()` en `Car` y ejecutará este código en su lugar. Cuando utilizas la herencia, puedes hacer que tus clases hijas retengan lo que necesites y anulen lo que no necesites de la clase padre.

Now if someone tries to call `fill_gas_tank()` with an electric car, Python will ignore the method `fill_gas_tank()` in `Car` and run this code instead. When you use inheritance, you can make your child classes retain what you need and override anything you don't need from the parent class.

Instancias como atributos (*Instances as Attributes*)

Al modelar algo del mundo real en código, puedes encontrarte con que vas añadiendo cada vez más detalles a una clase. Te darás cuenta de que tienes una lista cada vez mayor de atributos y métodos y que tus archivos son cada vez más largos. En estas situaciones, puedes reconocer que parte de una clase puede escribirse como una clase independiente. Puedes dividir tu clase grande en clases más pequeñas que funcionen juntas; este enfoque se denomina *composition*.

When modeling something from the real world in code, you may find that you're adding more and more detail to a class. You'll find that you have a growing list of attributes and methods and that your files are becoming lengthy. In these situations, you might recognize that part of one class can be written as a separate class. You can break your large class into smaller classes that work together; this approach is called *composition* .

Por ejemplo, si seguimos añadiendo detalles a la clase `ElectricCar` , podríamos darnos cuenta de que estamos añadiendo muchos atributos y métodos específicos de la batería del coche. Cuando veamos que esto ocurre, podemos detenernos y trasladar esos atributos y métodos a una clase independiente llamada `Battery` . Entonces podremos utilizar una instancia de `Battery` como atributo en la clase `ElectricCar` :

For example, if we continue adding detail to the `ElectricCar` class, we might notice that we're adding many attributes and methods specific to the car's battery. When we see this happening, we can stop and move those attributes and methods to a separate class called `Battery` . Then we can use a `Battery` instance as an attribute in the `ElectricCar` class:

```
class Car:
    --snip--

class Battery:
    """A simple attempt to model a battery for an electric car."""

    ❶ def __init__(self, battery_size=40):
        """Initialize the battery's attributes."""
        self.battery_size = battery_size

    ❷ def describe_battery(self):
        """Print a statement describing the battery size."""
        print(f"This car has a {self.battery_size}-kWh battery.")

class ElectricCar(Car):
    """Represent aspects of a car, specific to electric vehicles."""

    def __init__(self, make, model, year):
        """
```

Initialize attributes of the parent class.
Then initialize attributes specific to an electric car.
"""

```
super().__init__(make, model, year)  
❸ self.battery = Battery()
```

```
my_leaf = ElectricCar('nissan', 'leaf', 2024)  
print(my_leaf.get_descriptive_name())  
my_leaf.battery.describe_battery()
```

Definimos una nueva clase llamada `Battery` que no hereda de ninguna otra clase. El método `__init__()` ❶ tiene un parámetro, `battery_size`, además de `self`. Se trata de un parámetro opcional que establece el tamaño de la pila en 40 si no se proporciona ningún valor. El método `describe_battery()` también se ha trasladado a esta clase ❷.

We define a new class called `Battery` that doesn't inherit from any other class. The `__init__()` method ❶ has one parameter, `battery_size`, in addition to `self`. This is an optional parameter that sets the battery's size to 40 if no value is provided. The method `describe_battery()` has been moved to this class as well ❷.

En la clase `ElectricCar`, añadimos ahora un atributo llamado `self.battery` ❸. Esta línea le dice a Python que cree una nueva instancia de `Battery` (con un tamaño por defecto de 40, porque no estamos especificando un valor) y asigne esa instancia al atributo `self.battery`. Esto ocurrirá cada vez que se llame al método `__init__()`; cualquier instancia de `ElectricCar` tendrá ahora una instancia de `Battery` creada automáticamente.

In the `ElectricCar` class, we now add an attribute called `self.battery` ❸. This line tells Python to create a new instance of `Battery` (with a default size of 40, because we're not specifying a value) and assign that instance to the attribute `self.battery`. This will happen every time the `__init__()` method is called; any `ElectricCar` instance will now have a `Battery` instance created automatically.

Creamos un coche eléctrico y lo asignamos a la variable `my_leaf`. Cuando queramos describir la batería, tendremos que trabajar a través del atributo `battery` del coche:

We create an electric car and assign it to the variable `my_leaf` . When we want to describe the battery, we need to work through the car's `battery` attribute:

```
my_leaf.battery.describe_battery()
```

Esta línea le dice a Python que busque la instancia `my_leaf` , encuentre su atributo `battery` y llame al método `describe_battery()` que está asociado a la instancia `Battery` asignada al atributo.

This line tells Python to look at the instance `my_leaf` , find its `battery` attribute, and call the method `describe_battery()` that's associated with the `Battery` instance assigned to the attribute.

El resultado es idéntico al que vimos anteriormente:

The output is identical to what we saw previously:

2024 Nissan Leaf

This car has a 40-kWh battery.

Esto parece mucho trabajo extra, pero ahora podemos describir la pila con tanto detalle como queramos sin saturar la clase `ElectricCar` . Añadamos otro método a `Battery` que informe de la autonomía del coche en función del tamaño de la batería:

This looks like a lot of extra work, but now we can describe the battery in as much detail as we want without cluttering the `ElectricCar` class. Let's add another method to `Battery` that reports the range of the car based on the battery size:

```
class Car:
```

```
    --snip--
```

```
class Battery:
```

```
    --snip--
```

```
    def get_range(self):
```

```

        """Print a statement about the range this battery provides."""
        if self.battery_size == 40:
            range = 150
        elif self.battery_size == 65:
            range = 225

        print(f"This car can go about {range} miles on a full charge.")

class ElectricCar(Car):
    --snip--

my_leaf = ElectricCar('nissan', 'leaf', 2024)
print(my_leaf.get_descriptive_name())
my_leaf.battery.describe_battery()
❶ my_leaf.battery.get_range()

```

El nuevo método `get_range()` realiza algunos análisis sencillos. Si la capacidad de la batería es de 40 kWh, `get_range()` establece la autonomía en 240 km, y si la capacidad es de 65 kWh, establece la autonomía en 225 km. A continuación, informa de este valor. Cuando queramos utilizar este método, tendremos que llamarlo de nuevo a través del atributo `battery` del coche ❶ .

The new method `get_range()` performs some simple analysis. If the battery's capacity is 40 kWh, `get_range()` sets the range to 150 miles, and if the capacity is 65 kWh, it sets the range to 225 miles. It then reports this value. When we want to use this method, we again have to call it through the car's `battery` attribute ❶ .

La salida nos indica la autonomía del coche en función del tamaño de su batería:

The output tells us the range of the car based on its battery size:

```

2024 Nissan Leaf
This car has a 40-kWh battery.
This car can go about 150 miles on a full charge.

```

Modelar objetos del mundo real (*Modeling Real-World Objects*)

Cuando empieces a modelar cosas más complicadas, como coches eléctricos, te plantearás cuestiones interesantes. ¿La autonomía de un coche eléctrico es una propiedad de la

batería o del coche? Si sólo estamos describiendo un coche, probablemente esté bien mantener la asociación del método `get_range()` con la clase `Battery` . Pero si estamos describiendo toda la línea de coches de un fabricante, probablemente queramos trasladar `get_range()` a la clase `ElectricCar` . El método `get_range()` seguiría comprobando el tamaño de la batería antes de determinar la autonomía, pero informaría de una autonomía específica para la clase de coche a la que está asociado. Alternativamente, podríamos mantener la asociación del método `get_range()` con la batería, pero pasándole un parámetro como `car_model` . El método `get_range()` informaría entonces de un rango basado en el tamaño de la batería y el modelo de coche.

As you begin to model more complicated things like electric cars, you'll wrestle with interesting questions. Is the range of an electric car a property of the battery or of the car? If we're only describing one car, it's probably fine to maintain the association of the method `get_range()` with the `Battery` class. But if we're describing a manufacturer's entire line of cars, we probably want to move `get_range()` to the `ElectricCar` class. The `get_range()` method would still check the battery size before determining the range, but it would report a range specific to the kind of car it's associated with. Alternatively, we could maintain the association of the `get_range()` method with the battery but pass it a parameter such as `car_model` . The `get_range()` method would then report a range based on the battery size and car model.

Esto te lleva a un punto interesante en tu crecimiento como programador. Cuando te enfrentas a preguntas como éstas, estás pensando a un nivel lógico superior en lugar de a un nivel centrado en la sintaxis. No estás pensando en Python, sino en cómo representar el mundo real en código. Cuando llegues a este punto, te darás cuenta de que a menudo no hay enfoques correctos o incorrectos para modelar situaciones del mundo real. Algunos enfoques son más eficientes que otros, pero se necesita práctica para encontrar las representaciones más eficientes. Si tu código funciona como quieres, ¡lo estás haciendo bien! No te desanimes si descubres que estás desmontando tus clases y reescribiéndolas varias veces utilizando distintos enfoques. En la búsqueda de un código preciso y eficiente, todo el mundo pasa por este proceso.

This brings you to an interesting point in your growth as a programmer. When you wrestle with questions like these, you're thinking at a higher logical level rather than a syntax-focused level. You're thinking not about Python, but about how to represent the real world in code. When you reach this point, you'll realize there are often no right or wrong approaches to modeling real-world situations. Some approaches are more efficient than others, but it takes practice to find the most efficient representations. If your code is working as you want it to, you're doing well! Don't be discouraged if you find you're ripping

apart your classes and rewriting them several times using different approaches. In the quest to write accurate, efficient code, everyone goes through this process.

Pruébalo tú mismo (*Try It Yourself*)

9-6. Puesto de helados: Un puesto de helados es una clase específica de restaurante. Escribe una clase llamada `IceCreamStand` que herede de la clase `Restaurant` que escribiste en el Ejercicio 9-1 (página 162) o en el Ejercicio 9-4 (página 166). Cualquiera de las dos versiones de la clase funcionará; simplemente elige la que más te guste. Añade un atributo llamado `flavors` que almacene una lista de sabores de helado. Escribe un método que muestre estos sabores. Crea una instancia de `IceCreamStand`, y llama a este método.

9-6. Ice Cream Stand: An ice cream stand is a specific kind of restaurant. Write a class called `IceCreamStand` that inherits from the `Restaurant` class you wrote in Exercise 9-1 (page 162) or Exercise 9-4 (page 166). Either version of the class will work; just pick the one you like better. Add an attribute called `flavors` that stores a list of ice cream flavors. Write a method that displays these flavors. Create an instance of `IceCreamStand`, and call this method.

9-7. Administrador: Un administrador es una clase especial de usuario. Escribe una clase llamada `Admin` que herede de la clase `User` que escribiste en el Ejercicio 9-3 (página 162) o en el Ejercicio 9-5 (página 167). Añade un atributo, `privileges`, que almacene una lista de cadenas como "can add post", "can delete post", "can ban user", etc. Escribe un método llamado `show_privileges()` que enumere el conjunto de privilegios del administrador. Crea una instancia de `Admin`, y llama a tu método.

9-7. Admin: An administrator is a special kind of user. Write a class called `Admin` that inherits from the `User` class you wrote in Exercise 9-3 (page 162) or Exercise 9-5 (page 167). Add an attribute, `privileges`, that stores a list of strings like "can add post", "can delete post", "can ban user", and so on. Write a method called `show_privileges()` that lists the administrator's set of privileges. Create an instance of `Admin`, and call your method.

9-8. Privilegios: Escribe una clase `Privileges` separada. La clase debe tener un atributo, `privileges`, que almacene una lista de cadenas como se describe en el Ejercicio 9-7. Traslada el método `show_privileges()` a esta clase. Crea una instancia de `Privileges` como atributo en la clase `Admin`. Crea una nueva instancia de `Admin` y utiliza su método para mostrar sus privilegios.

9-8. Privileges: Write a separate `Privileges` class. The class should have one attribute, `privileges`, that stores a list of strings as described in Exercise 9-7. Move the `show_privileges()` method to this class. Make a `Privileges` instance as an attribute in the `Admin` class. Create a new instance of `Admin` and use your method to show its privileges.

9-9. Actualización de la Batería: Utiliza la versión final de *electric_car.py* de esta sección. Añade un método a la clase `Battery` llamado `upgrade_battery()`. Este método debe comprobar el tamaño de la batería y establecer la capacidad en 65 si aún no lo está. Crea un coche eléctrico con un tamaño de batería por defecto, llama a `get_range()` una vez y, a continuación, llama a `get_range()` una segunda vez después de actualizar la batería. Deberías ver un aumento de la autonomía del coche.

9-9. Battery Upgrade: Use the final version of *electric_car.py* from this section. Add a method to the `Battery` class called `upgrade_battery()`. This method should check the battery size and set the capacity to 65 if it isn't already. Make an electric car with a default battery size, call `get_range()` once, and then call `get_range()` a second time after upgrading the battery. You should see an increase in the car's range.

Importar clases (*Importing Classes*)

A medida que añades más funcionalidad a tus clases, tus archivos pueden hacerse largos, incluso cuando utilizas la herencia y la composición adecuadamente. De acuerdo con la filosofía general de Python, querrás mantener tus archivos lo más despejados posible. Para ayudarte, Python te permite almacenar clases en módulos y luego importar las clases que necesites en tu programa principal.

As you add more functionality to your classes, your files can get long, even when you use inheritance and composition properly. In keeping with the overall philosophy of Python, you'll want to keep your files as uncluttered as possible. To help, Python lets you store classes in modules and then import the classes you need into your main program.

Importar una sola clase (*Importing a Single Class*)

Creemos un módulo que contenga sólo la clase `Car` . Esto plantea un sutil problema de nomenclatura: ya tenemos un archivo llamado `car.py` en este capítulo, pero este módulo debería llamarse `car.py` porque contiene código que representa un coche. Resolveremos este problema de nomenclatura almacenando la clase `Car` en un módulo llamado `car.py` , en sustitución del archivo `car.py` que utilizábamos anteriormente. A partir de ahora, cualquier programa que utilice este módulo necesitará un nombre de archivo más específico, como `my_car.py` . Aquí tienes `car.py` con sólo el código de la clase `Car` :

Let's create a module containing just the `Car` class. This brings up a subtle naming issue: we already have a file named `car.py` in this chapter, but this module should be named `car.py` because it contains code representing a car. We'll resolve this naming issue by storing the `Car` class in a module named `car.py` , replacing the `car.py` file we were previously using. From now on, any program that uses this module will need a more specific filename, such as `my_car.py` . Here's `car.py` with just the code from the class `Car` :

coche.py

car.py

```
❶ """A class that can be used to represent a car."""
```

```
class Car:
```

```
    """A simple attempt to represent a car."""
```

```
    def __init__(self, make, model, year):
```

```
        """Initialize attributes to describe a car."""
```

```
        self.make = make
```

```
        self.model = model
```

```
        self.year = year
```

```
        self.odometer_reading = 0
```

```
    def get_descriptive_name(self):
```

```
        """Return a neatly formatted descriptive name."""
```

```
        long_name = f'{self.year} {self.make} {self.model}'
```

```
        return long_name.title()
```



```

def read_odometer(self):
    """Print a statement showing the car's mileage."""
    print(f"This car has {self.odometer_reading} miles on it.")

def update_odometer(self, mileage):
    """
    Set the odometer reading to the given value.
    Reject the change if it attempts to roll the odometer back.
    """
    if mileage >= self.odometer_reading:
        self.odometer_reading = mileage
    else:
        print("You can't roll back an odometer!")

def increment_odometer(self, miles):
    """Add the given amount to the odometer reading."""
    self.odometer_reading += miles

```

Incluimos un docstring a nivel de módulo que describe brevemente el contenido de este módulo ❶ . Debes escribir un docstring para cada módulo que crees.

We include a module-level docstring that briefly describes the contents of this module ❶ . You should write a docstring for each module you create.

Ahora creamos un archivo independiente llamado *my_car.py* . Este archivo importará la clase `Car` y luego creará una instancia a partir de esa clase:

Now we make a separate file called *my_car.py* . This file will import the `Car` class and then create an instance from that class:

mi_coche.py

my_car.py

❶ from car import Car

```
my_new_car = Car('audi', 'a4', 2024)
print(my_new_car.get_descriptive_name())
```

```
my_new_car.odometer_reading = 23
my_new_car.read_odometer()
```

La sentencia `import` ❶ indica a Python que abra el módulo `car` e importe la clase `Car`. Ahora podemos utilizar la clase `Car` como si estuviera definida en este archivo. El resultado es el mismo que hemos visto antes:

The `import` statement ❶ tells Python to open the `car` module and import the class `Car`. Now we can use the `Car` class as if it were defined in this file. The output is the same as we saw earlier:

2024 Audi A4

This car has 23 miles on it.

Importar clases es una forma eficaz de programar. Imagina lo largo que sería este archivo de programa si se incluyera toda la clase `Car`. En cambio, si trasladas la clase a un módulo e importas el módulo, sigues obteniendo toda la misma funcionalidad, pero mantienes tu archivo de programa principal limpio y fácil de leer. También almacenas la mayor parte de la lógica en archivos separados; una vez que tus clases funcionen como quieres, puedes dejar esos archivos tranquilos y centrarte en la lógica de alto nivel de tu programa principal.

Importing classes is an effective way to program. Picture how long this program file would be if the entire `Car` class were included. When you instead move the class to a module and import the module, you still get all the same functionality, but you keep your main program file clean and easy to read. You also store most of the logic in separate files; once your classes work as you want them to, you can leave those files alone and focus on the higher-level logic of your main program.

Almacenar varias clases en un módulo *(Storing Multiple Classes in a Module)*

Puedes almacenar tantas clases como necesites en un solo módulo, aunque cada clase de un módulo debe estar relacionada de alguna manera. Las clases `Battery` y `ElectricCar` ayudan ambas a representar coches, así que añadámoslas al módulo `car.py`.

You can store as many classes as you need in a single module, although each class in a module should be related somehow. The classes `Battery` and `ElectricCar` both help represent cars, so let's add them to the module *car.py*.

coche.py

car.py

```
"""A set of classes used to represent gas and electric cars."""
```

```
class Car:
```

```
    --snip--
```

```
class Battery:
```

```
    """A simple attempt to model a battery for an electric car."""
```

```
    def __init__(self, battery_size=40):
```

```
        """Initialize the battery's attributes."""
```

```
        self.battery_size = battery_size
```

```
    def describe_battery(self):
```

```
        """Print a statement describing the battery size."""
```

```
        print(f"This car has a {self.battery_size}-kWh battery.")
```

```
    def get_range(self):
```

```
        """Print a statement about the range this battery provides."""
```

```
        if self.battery_size == 40:
```

```
            range = 150
```

```
        elif self.battery_size == 65:
```

```
            range = 225
```

```
        print(f"This car can go about {range} miles on a full charge.")
```

```
class ElectricCar(Car):
```

```
    """Models aspects of a car, specific to electric vehicles."""
```

```
    def __init__(self, make, model, year):
```

```
"""
Initialize attributes of the parent class.
Then initialize attributes specific to an electric car.
"""
super().__init__(make, model, year)
self.battery = Battery()
```

Ahora podemos crear un nuevo archivo llamado *my_electric_car.py*, importar la clase `ElectricCar` y crear un coche eléctrico:

Now we can make a new file called *my_electric_car.py*, import the `ElectricCar` class, and make an electric car:

mi_coche_eléctrico.py

my_electric_car.py

```
from car import ElectricCar

my_leaf = ElectricCar('nissan', 'leaf', 2024)
print(my_leaf.get_descriptive_name())
my_leaf.battery.describe_battery()
my_leaf.battery.get_range()
```

Esto tiene el mismo resultado que vimos antes, aunque la mayor parte de la lógica esté oculta en un módulo:

This has the same output we saw earlier, even though most of the logic is hidden away in a module:

2024 Nissan Leaf
This car has a 40-kWh battery.
This car can go about 150 miles on a full charge.

Importar varias clases de un módulo *(Importing Multiple Classes from a Module)*

Puedes importar tantas clases como necesites en un archivo de programa. Si queremos hacer un coche normal y un coche eléctrico en el mismo archivo, tenemos que importar ambas clases, `Car` y `ElectricCar` :

You can import as many classes as you need into a program file. If we want to make a regular car and an electric car in the same file, we need to import both classes, `Car` and `ElectricCar` :

mis_coches.py

my_cars.py

```
❶ from car import Car, ElectricCar

❷ my_mustang = Car('ford', 'mustang', 2024)
print(my_mustang.get_descriptive_name())
❸ my_leaf = ElectricCar('nissan', 'leaf', 2024)
print(my_leaf.get_descriptive_name())
```

Puedes importar varias clases de un módulo separando cada clase con una coma ❶ . Una vez que hayas importado las clases necesarias, puedes crear tantas instancias de cada clase como necesites.

You import multiple classes from a module by separating each class with a comma ❶ . Once you've imported the necessary classes, you're free to make as many instances of each class as you need.

En este ejemplo creamos un Ford Mustang de gasolina ❷ y luego un Nissan Leaf eléctrico ❸ :

In this example we make a gas-powered Ford Mustang ❷ and then an electric Nissan Leaf ❸ :

2024 Ford Mustang
2024 Nissan Leaf

Importar un módulo entero (*Importing an Entire Module*)

También puedes importar un módulo entero y acceder a las clases que necesites utilizando la notación por puntos. Este enfoque es sencillo y da como resultado un código fácil de leer. Como cada llamada que crea una instancia de una clase incluye el nombre del módulo, no tendrás conflictos de nombres con los nombres utilizados en el archivo actual.

You can also import an entire module and then access the classes you need using dot notation. This approach is simple and results in code that is easy to read. Because every call that creates an instance of a class includes the module name, you won't have naming conflicts with any names used in the current file.

Este es el aspecto que tiene importar todo el módulo `car` y luego crear un coche normal y un coche eléctrico:

Here's what it looks like to import the entire `car` module and then create a regular car and an electric car:

mis_coches.py

my_cars.py

```
❶ import car

❷ my_mustang = car.Car('ford', 'mustang', 2024)
print(my_mustang.get_descriptive_name())

❸ my_leaf = car.ElectricCar('nissan', 'leaf', 2024)
print(my_leaf.get_descriptive_name())
```

Primero importamos todo el módulo `car` ❶. A continuación, accedemos a las clases que necesitamos mediante la sintaxis `module_name . ClassName`. Creamos de nuevo un Ford

Mustang ❷ , y un Nissan Leaf ❸ .

First we import the entire `car` module ❶ . We then access the classes we need through the `module_name . ClassName` syntax. We again create a Ford Mustang ❷ , and a Nissan Leaf ❸ .

Importar todas las clases de un módulo *(Importing All Classes from a Module)*

Puedes importar todas las clases de un módulo utilizando la siguiente sintaxis:

You can import every class from a module using the following syntax:

```
from module_name import *
```

Este método no es recomendable por dos razones. En primer lugar, es útil poder leer las declaraciones `import` al principio de un archivo y tener una idea clara de qué clases utiliza un programa. Con este enfoque no está claro qué clases estás utilizando del módulo. Este enfoque también puede llevar a confusiones con los nombres en el archivo. Si importas accidentalmente una clase con el mismo nombre que otra en el archivo de tu programa, puedes crear errores difíciles de diagnosticar. Muestro esto aquí porque, aunque no es un enfoque recomendado, es probable que lo veas en el código de otras personas en algún momento.

This method is not recommended for two reasons. First, it's helpful to be able to read the `import` statements at the top of a file and get a clear sense of which classes a program uses. With this approach it's unclear which classes you're using from the module. This approach can also lead to confusion with names in the file. If you accidentally import a class with the same name as something else in your program file, you can create errors that are hard to diagnose. I show this here because even though it's not a recommended approach, you're likely to see it in other people's code at some point.

Si necesitas importar muchas clases de un módulo, es mejor que importes todo el módulo y utilices la sintaxis `module_name.ClassName` . No verás todas las clases utilizadas en la parte superior del archivo, pero verás claramente dónde se utiliza el módulo en el programa. También evitarás los posibles conflictos de nombres que pueden surgir cuando importas todas las clases de un módulo.

If you need to import many classes from a module, you're better off importing the entire module and using the `module_name.ClassName` syntax. You won't see all the classes used at the top of the file, but you'll see clearly where the module is used in the program. You'll also avoid the potential naming conflicts that can arise when you import every class in a module.

Importar un módulo dentro de otro módulo (*Importing a Module into a Module*)

A veces querrás distribuir tus clases en varios módulos para evitar que un archivo crezca demasiado y para evitar almacenar clases no relacionadas en el mismo módulo. Cuando almacenas tus clases en varios módulos, puedes encontrar que una clase de un módulo depende de una clase de otro módulo. Cuando esto ocurra, puedes importar la clase necesaria en el primer módulo.

Sometimes you'll want to spread out your classes over several modules to keep any one file from growing too large and avoid storing unrelated classes in the same module. When you store your classes in several modules, you may find that a class in one module depends on a class in another module. When this happens, you can import the required class into the first module.

Por ejemplo, almacenemos la clase `Car` en un módulo y las clases `ElectricCar` y `Battery` en otro módulo. Crearemos un nuevo módulo llamado *electric_car.py* -reemplazando el archivo *electric_car.py* que creamos antes- y copiaremos sólo las clases `Battery` y `ElectricCar` en este archivo:

For example, let's store the `Car` class in one module and the `ElectricCar` and `Battery` classes in a separate module. We'll make a new module called *electric_car.py* —replacing the *electric_car.py* file we created earlier—and copy just the `Battery` and `ElectricCar` classes into this file:

coche_eléctrico.py

electric_car.py

```
"""A set of classes that can be used to represent electric cars."""
```

```
from car import Car
```

```
class Battery:  
    --snip--
```

```
class ElectricCar(Car):  
    --snip--
```

La clase `ElectricCar` necesita acceder a su clase padre `Car` , así que importaremos `Car` directamente al módulo. Si olvidamos esta línea, Python emitirá un error cuando intentemos importar el módulo `electric_car` . También tenemos que actualizar el módulo `Car` para que sólo contenga la clase `Car` :

The class `ElectricCar` needs access to its parent class `Car` , so we import `Car` directly into the module. If we forget this line, Python will raise an error when we try to import the `electric_car` module. We also need to update the `Car` module so it contains only the `Car` class:

coche.py

car.py

```
"""A class that can be used to represent a car."""
```

```
class Car:  
    --snip--
```

Ahora podemos importar de cada módulo por separado y crear la clase de coche que necesitemos:

Now we can import from each module separately and create whatever kind of car we need:

mis_coches.py

my_cars.py

```
from car import Car
from electric_car import ElectricCar

my_mustang = Car('ford', 'mustang', 2024)
print(my_mustang.get_descriptive_name())

my_leaf = ElectricCar('nissan', 'leaf', 2024)
print(my_leaf.get_descriptive_name())
```

Importamos `Car` de su módulo, y `ElectricCar` de su módulo. A continuación, creamos un coche normal y un coche eléctrico. Ambos coches se crean correctamente:

We import `Car` from its module, and `ElectricCar` from its module. We then create one regular car and one electric car. Both cars are created correctly:

```
2024 Ford Mustang
2024 Nissan Leaf
```

Utilizar alias (*Using Aliases*)

Como viste en el Capítulo 8 , los alias pueden ser muy útiles cuando utilizas módulos para organizar el código de tus proyectos. También puedes utilizar alias al importar clases.

As you saw in Chapter 8 , aliases can be quite helpful when using modules to organize your projects' code. You can use aliases when importing classes as well.

Como ejemplo, considera un programa en el que quieras fabricar un montón de coches eléctricos. Podría resultar tedioso escribir (y leer) `ElectricCar` una y otra vez. Puedes dar a `ElectricCar` un alias en la sentencia `import`:

As an example, consider a program where you want to make a bunch of electric cars. It might get tedious to type (and read) `ElectricCar` over and over again. You can give `ElectricCar` an alias in the import statement:

```
from electric_car import ElectricCar as EC
```

Ahora puedes utilizar este alias siempre que quieras fabricar un coche eléctrico:

Now you can use this alias whenever you want to make an electric car:

```
my_leaf = EC('nissan', 'leaf', 2024)
```

También puedes dar un alias a un módulo. Aquí tienes cómo importar todo el módulo `electric_car` utilizando un alias:

You can also give a module an alias. Here's how to import the entire `electric_car` module using an alias:

```
import electric_car as ec
```

Ahora puedes utilizar este alias de módulo con el nombre completo de la clase:

Now you can use this module alias with the full class name:

```
my_leaf = ec.ElectricCar('nissan', 'leaf', 2024)
```

Encontrar tu propio flujo de trabajo (*Finding Your Own Workflow*)

Como puedes ver, Python te da muchas opciones sobre cómo estructurar el código en un proyecto grande. Es importante que conozcas todas estas posibilidades para que puedas determinar las mejores formas de organizar tus proyectos, así como entender los proyectos de otras personas.

As you can see, Python gives you many options for how to structure code in a large project. It's important to know all these possibilities so you can determine the best ways to organize your projects as well as understand other people's projects.

Cuando estés empezando, mantén una estructura de código sencilla. Intenta hacerlo todo en un solo archivo y mover tus clases a módulos separados una vez que todo funcione. Si te gusta cómo interactúan los módulos y los archivos, intenta almacenar tus clases en módulos cuando empieces un proyecto. Encuentra un enfoque que te permita escribir código que funcione, y sigue a partir de ahí.

When you're starting out, keep your code structure simple. Try doing everything in one file and moving your classes to separate modules once everything is working. If you like how modules and files interact, try storing your classes in modules when you start a project. Find an approach that lets you write code that works, and go from there.

Pruébalo tú mismo *(Try It Yourself)*

9-10. Restaurante importado: Utilizando tu última clase `Restaurant`, almacénala en un módulo. Crea un archivo aparte que importe `Restaurant`. Crea una instancia de `Restaurant` y llama a uno de los métodos de `Restaurant` para demostrar que la sentencia `import` funciona correctamente.

9-10. Imported Restaurant: Using your latest `Restaurant` class, store it in a module. Make a separate file that imports `Restaurant`. Make a `Restaurant` instance, and call one of `Restaurant`'s methods to show that the `import` statement is working properly.

9-11. Admin. importado: Empieza con tu trabajo del Ejercicio 9-8 (página 173). Guarda las clases `User`, `Privileges`, y `Admin` en un módulo. Crea un archivo separado, crea una instancia `Admin` y llama a `show_privileges()` para mostrar que todo funciona correctamente.

9-11. Imported Admin: Start with your work from Exercise 9-8 (page 173). Store the classes `User`, `Privileges`, and `Admin` in one module. Create a separate file, make an `Admin` instance, and call `show_privileges()` to show that everything is working correctly.

9-12. Módulos múltiples: Almacena la clase `User` en un módulo, y almacena las clases `Privileges` y `Admin` en un módulo aparte. En un archivo aparte, crea una instancia de `Admin` y llama a `show_privileges()` para demostrar que todo sigue funcionando correctamente.

9-12. Multiple Modules: Store the `User` class in one module, and store the `Privileges` and `Admin` classes in a separate module. In a separate file, create an `Admin` instance and call `show_privileges()` to show that everything is still working correctly.

La biblioteca estándar de Python (*The Python Standard Library*)

La *Python standard library* es un conjunto de módulos incluidos en cada instalación de Python. Ahora que tienes una comprensión básica de cómo funcionan las funciones y las clases, puedes empezar a utilizar módulos como éstos que han escrito otros programadores. Puedes utilizar cualquier función o clase de la biblioteca estándar incluyendo una simple sentencia `import` al principio de tu archivo. Veamos un módulo, `random`, que puede ser útil para modelar muchas situaciones del mundo real.

The *Python standard library* is a set of modules included with every Python installation. Now that you have a basic understanding of how functions and classes work, you can start to use modules like these that other programmers have written. You can use any function or class in the standard library by including a simple `import` statement at the top of your file. Let's look at one module, `random`, which can be useful in modeling many real-world situations.

Una función interesante del módulo aleatorio es `randint()`. Esta función toma dos argumentos enteros y devuelve un número entero seleccionado aleatoriamente entre esos números (e incluyéndolos).

One interesting function from the `random` module is `randint()`. This function takes two integer arguments and returns a randomly selected integer between (and including) those numbers.

He aquí cómo generar un número aleatorio entre 1 y 6:

Here's how to generate a random number between 1 and 6:

```
>>> from random import randint
>>> randint(1, 6)
3
```

Otra función útil es `choice()` . Esta función toma una lista o tupla y devuelve un elemento elegido al azar:

Another useful function is `choice()` . This function takes in a list or tuple and returns a randomly chosen element:

```
>>> from random import choice
>>> players = ['charles', 'martina', 'michael', 'florence', 'eli']
>>> first_up = choice(players)
>>> first_up
'florence'
```

El módulo `random` no debería utilizarse para crear aplicaciones relacionadas con la seguridad, pero funciona bien para muchos proyectos divertidos e interesantes.

The `random` module shouldn't be used when building security-related applications, but it works well for many fun and interesting projects.

Nota (Note)

También puedes descargar módulos de fuentes externas. Verás varios de estos ejemplos en la Parte II, donde necesitaremos módulos externos para completar cada proyecto.

You can also download modules from external sources. You'll see a number of these examples in Part II, where we'll need external modules to complete each project.

Pruébalo tú mismo (*Try It Yourself*)

9-13. Dados: Crea una clase `Die` con un atributo llamado `sides` , que tiene un valor por defecto de 6. Escribe un método llamado `roll_die()` que imprima un número aleatorio entre 1 y el número de caras que tiene el dado. Haz un dado de 6 caras y lánzalo 10 veces.

9-13. Dice: Make a class `Die` with one attribute called `sides` , which has a default value of 6. Write a method called `roll_die()` that prints a random number between 1 and the number of sides the die has. Make a 6-sided die and roll it 10 times.

Crea un dado de 10 caras y otro de 20 caras. Tira cada dado 10 veces.

Make a 10-sided die and a 20-sided die. Roll each die 10 times.

9-14. Lotería: Haz una lista o tupla que contenga una serie de 10 números y 5 letras. Selecciona al azar 4 números o letras de la lista e imprime un mensaje diciendo que cualquier boleto que coincida con esos 4 números o letras gana un premio.

9-14. Lottery: Make a list or tuple containing a series of 10 numbers and 5 letters. Randomly select 4 numbers or letters from the list and print a message saying that any ticket matching these 4 numbers or letters wins a prize.

9-15. Análisis de la Lotería: Puedes utilizar un bucle para ver lo difícil que puede ser ganar el tipo de lotería que acabas de modelar. Haz una lista o tupla llamada `my_ticket` . Escribe un bucle que siga sacando números hasta que gane el boleto. Imprime un mensaje informando de cuántas veces ha tenido que ejecutarse el bucle para darte un boleto ganador.

9-15. Lottery Analysis: You can use a loop to see how hard it might be to win the kind of lottery you just modeled. Make a list or tuple called `my_ticket` . Write a loop that keeps

pulling numbers until your ticket wins. Print a message reporting how many times the loop had to run to give you a winning ticket.

9-16. Módulo Python de la Semana: Un recurso excelente para explorar la biblioteca estándar de Python es un sitio llamado *Python Module of the Week* . Entra en <https://pymotw.com> y mira la tabla de contenidos. Busca un módulo que te parezca interesante y lee sobre él, quizás empezando por el módulo `random` .

9-16. Python Module of the Week: One excellent resource for exploring the Python standard library is a site called *Python Module of the Week* . Go to <https://pymotw.com> and look at the table of contents. Find a module that looks interesting to you and read about it, perhaps starting with the `random` module.

Clases de estilo (*Styling Classes*)

Merece la pena aclarar algunas cuestiones de estilo relacionadas con las clases, especialmente a medida que tus programas se complican.

A few styling issues related to classes are worth clarifying, especially as your programs become more complicated.

Los nombres de las clases deben escribirse en *CamelCase* . Para ello, escribe en mayúscula la primera letra de cada palabra del nombre y no utilices guiones bajos. Los nombres de instancias y módulos deben escribirse en minúsculas, con guiones bajos entre las palabras.

Class names should be written in *CamelCase* . To do this, capitalize the first letter of each word in the name, and don't use underscores. Instance and module names should be written in lowercase, with underscores between words.

Cada clase debe tener un docstring inmediatamente después de la definición de la clase. La docstring debe ser una breve descripción de lo que hace la clase, y debes seguir las mismas convenciones de formato que utilizaste para escribir docstrings en funciones. Cada módulo debe tener también una docstring que describa para qué pueden utilizarse las clases de un módulo.

Every class should have a docstring immediately following the class definition. The docstring should be a brief description of what the class does, and you should follow the same formatting conventions you used for writing docstrings in functions. Each module should also have a docstring describing what the classes in a module can be used for.

Puedes utilizar líneas en blanco para organizar el código, pero no las utilices en exceso. Dentro de una clase puedes utilizar una línea en blanco entre métodos, y dentro de un módulo puedes utilizar dos líneas en blanco para separar clases.

You can use blank lines to organize code, but don't use them excessively. Within a class you can use one blank line between methods, and within a module you can use two blank lines to separate classes.

Si necesitas importar un módulo de la biblioteca estándar y un módulo escrito por ti, coloca primero la sentencia `import` del módulo de la biblioteca estándar. A continuación, añade una línea en blanco y la sentencia `import` del módulo que has escrito. En programas con varias declaraciones de importación, esta convención facilita ver de dónde proceden los distintos módulos utilizados en el programa.

If you need to import a module from the standard library and a module that you wrote, place the import statement for the standard library module first. Then add a blank line and the import statement for the module you wrote. In programs with multiple import statements, this convention makes it easier to see where the different modules used in the program come from.

Resumen (Summary)

En este capítulo has aprendido a escribir tus propias clases. Aprendiste a almacenar información en una clase utilizando atributos y a escribir métodos que den a tus clases el comportamiento que necesitan. Aprendiste a escribir métodos `__init__()` que crean instancias a partir de tus clases con exactamente los atributos que deseas. Viste cómo modificar los atributos de una instancia directamente y a través de métodos. Aprendiste que la herencia puede simplificar la creación de clases relacionadas entre sí, y aprendiste a utilizar instancias de una clase como atributos en otra clase para mantener la sencillez de cada clase.

In this chapter, you learned how to write your own classes. You learned how to store information in a class using attributes and how to write methods that give your classes the behavior they need. You learned to write `__init__()` methods that create instances from your classes with exactly the attributes you want. You saw how to modify the attributes of an instance directly and through methods. You learned that inheritance can simplify the creation of classes that are related to each other, and you learned to use instances of one class as attributes in another class to keep each class simple.

Viste cómo almacenar las clases en módulos e importar las clases que necesitas en los archivos donde se utilizarán puede mantener organizados tus proyectos. Empezaste a aprender sobre la biblioteca estándar de Python, y viste un ejemplo basado en el módulo `random`. Por último, aprendiste a dar estilo a tus clases utilizando las convenciones de Python.

You saw how storing classes in modules and importing classes you need into the files where they'll be used can keep your projects organized. You started learning about the Python standard library, and you saw an example based on the `random` module. Finally, you learned to style your classes using Python conventions.

En el Capítulo 10, aprenderás a trabajar con archivos para poder guardar el trabajo que has hecho en un programa y el trabajo que has permitido hacer a los usuarios. También aprenderás sobre *exceptions*, una clase especial de Python diseñada para ayudarte a responder a los errores cuando surjan.

In Chapter 10, you'll learn to work with files so you can save the work you've done in a program and the work you've allowed users to do. You'll also learn about *exceptions*, a special Python class designed to help you respond to errors when they arise.

Capítulo 10: Archivos y excepciones (Chapter 10: Files and Exceptions)

10

Archivos y excepciones (10 Files and Exceptions)



DESCRIPCIÓN DE LA
IMAGEN (En español):

La imagen muestra una ilustración en blanco y negro de una serpiente con expresión sonriente y ojos con gafas. La serpiente está envuelta alrededor de un motor a reacción, sugiriendo velocidad y poder. El fondo es gris, proporcionando contraste.

Ahora que dominas las habilidades básicas que necesitas para escribir programas organizados y fáciles de usar, es hora de pensar en hacer que tus programas sean aún más relevantes y utilizables. En este capítulo, aprenderás a trabajar con archivos para que tus programas puedan analizar rápidamente muchos datos.

Now that you've mastered the basic skills you need to write organized programs that are easy to use, it's time to think about making your programs even more relevant and usable. In this chapter, you'll learn to work with files so your programs can quickly analyze lots of data.

Aprenderás a gestionar errores para que tus programas no se bloqueen cuando se encuentren con situaciones inesperadas. Aprenderás sobre *exceptions*, que son objetos especiales que Python crea para gestionar los errores que surgen mientras se ejecuta un programa. También conocerás el módulo `json`, que te permite guardar los datos del usuario para que no se pierdan cuando tu programa deje de ejecutarse.

You'll learn to handle errors so your programs don't crash when they encounter unexpected situations. You'll learn about *exceptions*, which are special objects Python

creates to manage errors that arise while a program is running. You'll also learn about the `json` module, which allows you to save user data so it isn't lost when your program stops running.

Aprender a trabajar con archivos y a guardar datos facilitará el uso de tus programas. Los usuarios podrán elegir qué datos introducir y cuándo hacerlo. La gente podrá ejecutar tu programa, hacer algún trabajo, y luego cerrar el programa y continuar donde lo dejaron. Aprender a manejar excepciones te ayudará a enfrentarte a situaciones en las que los archivos no existen y a lidiar con otros problemas que pueden hacer que tus programas se bloqueen. Esto hará que tus programas sean más robustos cuando se encuentren con datos erróneos, tanto si proceden de errores inocentes de como de intentos maliciosos de romper tus programas. Con las habilidades que aprenderás en este capítulo, conseguirás que tus programas sean más aplicables, utilizables y estables.

Learning to work with files and save data will make your programs easier for people to use. Users will be able to choose what data to enter and when to enter it. People will be able to run your program, do some work, and then close the program and pick up where they left off. Learning to handle exceptions will help you deal with situations in which files don't exist and deal with other problems that can cause your programs to crash. This will make your programs more robust when they encounter bad data, whether it comes from innocent mistakes or from malicious attempts to break your programs. With the skills you'll learn in this chapter, you'll make your programs more applicable, usable, and stable.

Lectura de un archivo (*Reading from a File*)

Los archivos de texto contienen una cantidad increíble de datos. Los archivos de texto pueden contener datos meteorológicos, datos de tráfico, datos socioeconómicos, obras literarias y mucho más. La lectura desde un archivo es especialmente útil en aplicaciones de análisis de datos, pero también es aplicable a cualquier situación en la que quieras analizar o modificar información almacenada en un archivo. Por ejemplo, puedes escribir un programa que lea el contenido de un archivo de texto y lo reescriba con un formato que permita visualizarlo en un navegador.

An incredible amount of data is available in text files. Text files can contain weather data, traffic data, socioeconomic data, literary works, and more. Reading from a file is particularly useful in data analysis applications, but it's also applicable to any situation in which you want to analyze or modify information stored in a file. For example, you can write a program that reads in the contents of a text file and rewrites the file with formatting that allows a browser to display it.

Cuando quieras trabajar con la información de un archivo de texto, el primer paso es leer el archivo en la memoria. A continuación, puedes trabajar con todo el contenido del archivo a la vez o línea por línea.

When you want to work with the information in a text file, the first step is to read the file into memory. You can then work through all of the file's contents at once or work through the contents line by line.

Leer el contenido de un fichero (*Reading the Contents of a File*)

Para empezar, necesitamos un archivo con algunas líneas de texto. Empecemos con un archivo que contenga *pi* con 30 decimales, con 10 decimales por línea:

To begin, we need a file with a few lines of text in it. Let's start with a file that contains *pi* to 30 decimal places, with 10 decimal places per line:

pi_digits.txt

pi_digits.txt

```
3.1415926535
8979323846
2643383279
```

Para probar tú mismo los siguientes ejemplos, puedes introducir estas líneas en un editor y guardar el archivo como *pi_digits.txt*, o puedes descargar el archivo de los recursos del libro a través de https://ehmatthes.github.io/pcc_3e. Guarda el archivo en el mismo directorio donde guardarás los programas de este capítulo.

To try the following examples yourself, you can enter these lines in an editor and save the file as *pi_digits.txt*, or you can download the file from the book's resources through https://ehmatthes.github.io/pcc_3e. Save the file in the same directory where you'll store this chapter's programs.

Aquí tienes un programa que abre este archivo, lo lee e imprime el contenido del archivo en la pantalla:

Here's a program that opens this file, reads it, and prints the contents of the file to the screen:

lector_archivo.py

file_reader.py

```
from pathlib import Path
```

```
❶ path = Path('pi_digits.txt')  
❷ contents = path.read_text()  
print(contents)
```

Para trabajar con el contenido de un archivo, tenemos que indicarle a Python la ruta al archivo. Un *path* es la ubicación exacta de un archivo o carpeta en un sistema. Python proporciona un módulo llamado `pathlib` que facilita el trabajo con archivos y directorios, independientemente del sistema operativo con el que tú o los usuarios de tu programa estéis trabajando. Un módulo que proporciona una funcionalidad específica como ésta suele denominarse *library*, de ahí el nombre `pathlib`.

To work with the contents of a file, we need to tell Python the path to the file. A *path* is the exact location of a file or folder on a system. Python provides a module called `pathlib` that makes it easier to work with files and directories, no matter which operating system you or your program's users are working with. A module that provides specific functionality like this is often called a *library*, hence the name `pathlib`.

Empezaremos importando la clase `Path` de `pathlib`. Hay muchas cosas que puedes hacer con un objeto `Path` que apunte a un archivo. Por ejemplo, puedes comprobar que el archivo existe antes de trabajar con él, leer su contenido o escribir nuevos datos en él. Aquí, construimos un objeto `Path` que representa el archivo *pi_digits.txt*, que asignamos a la variable `path` ❶. Como este archivo está guardado en el mismo directorio que el

archivo `.py` que estamos escribiendo, el nombre del archivo es todo lo que `Path` necesita para acceder al archivo.

We start by importing the `Path` class from `pathlib`. There's a lot you can do with a `Path` object that points to a file. For example, you can check that the file exists before working with it, read the file's contents, or write new data to the file. Here, we build a `Path` object representing the file `pi_digits.txt`, which we assign to the variable `path` ❶. Since this file is saved in the same directory as the `.py` file we're writing, the filename is all that `Path` needs to access the file.

Nota (Note)

VS Code busca los archivos en la carpeta que se abrió más recientemente. Si utilizas VS Code, empieza abriendo la carpeta donde guardas los programas de este capítulo. Por ejemplo, si estás guardando los archivos de tus programas en una carpeta llamada `chapter_10`, pulsa CTRL-O (⌘-O en macOS), y abre esa carpeta.

VS Code looks for files in the folder that was most recently opened. If you're using VS Code, start by opening the folder where you're storing this chapter's programs. For example, if you're saving your program files in a folder called `chapter_10`, press CTRL-O (⌘-O on macOS), and open that folder.

Una vez que tenemos un objeto `Path` que representa a `pi_digits.txt`, utilizamos el método `read_text()` para leer todo el contenido del archivo ❷. El contenido del archivo se devuelve como una única cadena, que asignamos a la variable `contents`. Cuando imprimimos el valor de `contents`, vemos todo el contenido del archivo de texto:

Once we have a `Path` object representing `pi_digits.txt`, we use the `read_text()` method to read the entire contents of the file ❷. The contents of the file are returned as a single string, which we assign to the variable `contents`. When we print the value of `contents`, we see the entire contents of the text file:

```
3.1415926535
8979323846
2643383279
```

La única diferencia entre esta salida y el archivo original es la línea en blanco adicional al final de la salida. La línea en blanco aparece porque `read_text()` devuelve una cadena vacía cuando llega al final del archivo; esta cadena vacía aparece como una línea en blanco.

The only difference between this output and the original file is the extra blank line at the end of the output. The blank line appears because `read_text()` returns an empty string when it reaches the end of the file; this empty string shows up as a blank line.

Podemos eliminar la línea en blanco adicional utilizando `rstrip()` en la cadena `contents` :

We can remove the extra blank line by using `rstrip()` on the `contents` string:

```
from pathlib import Path

path = Path('pi_digits.txt')
contents = path.read_text()
contents = contents.rstrip()
print(contents)
```

Recuerda del Capítulo 2 que el método `rstrip()` de Python elimina, o quita, cualquier carácter de espacio en blanco de la parte derecha de una cadena. Ahora la salida coincide exactamente con el contenido del archivo original:

Recall from Chapter 2 that Python's `rstrip()` method removes, or strips, any whitespace characters from the right side of a string. Now the output matches the contents of the original file exactly:

```
3.1415926535
8979323846
2643383279
```


Podemos eliminar el carácter de nueva línea final cuando leamos el contenido del archivo, aplicando el método `rstrip()` inmediatamente después de llamar a `read_text()` :

We can strip the trailing newline character when we read the contents of the file, by applying the `rstrip()` method immediately after calling `read_text()` :

```
contents = path.read_text().rstrip()
```

Esta línea indica a Python que llame al método `read_text()` en el archivo con el que estamos trabajando. Luego aplica el método `rstrip()` a la cadena que devuelve `read_text()` . La cadena limpiada se asigna entonces a la variable `contents` . Este método se denomina *method chaining* , y lo verás utilizado a menudo en programación.

This line tells Python to call the `read_text()` method on the file we're working with. Then it applies the `rstrip()` method to the string that `read_text()` returns. The cleaned-up string is then assigned to the variable `contents` . This approach is called *method chaining* , and you'll see it used often in programming.

Rutas de archivos relativas y absolutas (*Relative and Absolute File Paths*)

Cuando pasas un nombre de archivo simple como `pi_digits.txt` a `Path` , Python busca en el directorio donde está almacenado el archivo que se está ejecutando en ese momento (es decir, tu archivo de programa `.py`).

When you pass a simple filename like `pi_digits.txt` to `Path` , Python looks in the directory where the file that's currently being executed (that is, your `.py` program file) is stored.

A veces, dependiendo de cómo organices tu trabajo, el archivo que quieres abrir no estará en el mismo directorio que tu archivo de programa. Por ejemplo, podrías almacenar tus archivos de programa en una carpeta llamada `python_work` ; dentro de `python_work` , podrías tener otra carpeta llamada `text_files` para distinguir tus archivos de programa de los archivos de texto que manipulan. Aunque `text_files` esté en `python_work` , pasar a `Path` el nombre de un archivo en `text_files` no funcionará, porque Python sólo buscará en `python_work` y se detendrá ahí; no seguirá y buscará en `text_files` . Para conseguir que Python abra archivos de un directorio distinto de aquel en el que está almacenado tu archivo de programa, necesitas proporcionar la ruta correcta.

Sometimes, depending on how you organize your work, the file you want to open won't be in the same directory as your program file. For example, you might store your program files in a folder called *python_work* ; inside *python_work* , you might have another folder called *text_files* to distinguish your program files from the text files they're manipulating. Even though *text_files* is in *python_work* , just passing `Path` the name of a file in *text_files* won't work, because Python will only look in *python_work* and stop there; it won't go on and look in *text_files*. To get Python to open files from a directory other than the one where your program file is stored, you need to provide the correct path.

Hay dos formas principales de especificar rutas en programación. Un *relative file path* le dice a Python que busque una ubicación determinada relativa al directorio donde está almacenado el archivo de programa que se está ejecutando en ese momento. Como *text_files* está dentro de *python_work* , necesitamos construir una ruta que empiece por el directorio *text_files* , y termine por el nombre del archivo. He aquí cómo construir esta ruta:

There are two main ways to specify paths in programming. A *relative file path* tells Python to look for a given location relative to the directory where the currently running program file is stored. Since *text_files* is inside *python_work* , we need to build a path that starts with the directory *text_files* , and ends with the filename. Here's how to build this path:

```
path = Path('text_files/filename.txt')
```

También puedes decirle a Python exactamente dónde está el archivo en tu ordenador, independientemente de dónde esté almacenado el programa que se está ejecutando. Esto se llama *absolute file path* . Puedes utilizar una ruta absoluta si una relativa no funciona. Por ejemplo, si has puesto *text_files* en alguna carpeta distinta de *python_work* , entonces pasar a `Path` la ruta `'text_files/ filename .txt'` no funcionará porque Python sólo buscará esa ubicación dentro de *python_work* . Tendrás que escribir una ruta absoluta para aclarar dónde quieres que busque Python.

You can also tell Python exactly where the file is on your computer, regardless of where the program that's being executed is stored. This is called an *absolute file path* . You can use an absolute path if a relative path doesn't work. For instance, if you've put *text_files* in some folder other than *python_work* , then just passing `Path` the path `'text_files/ filename .txt'` won't work because Python will only look for that location inside *python_work* . You'll need to write out an absolute path to clarify where you want Python to look.

Las rutas absolutas suelen ser más largas que las relativas, porque empiezan en la carpeta raíz de tu sistema:

Absolute paths are usually longer than relative paths, because they start at your system's root folder:

```
path = Path('/home/eric/data_files/text_files/filename.txt')
```

Utilizando rutas absolutas, puedes leer archivos desde cualquier ubicación de tu sistema. Por ahora, lo más fácil es almacenar los archivos en el mismo directorio que los archivos de tu programa, o en una carpeta como *text_files* dentro del directorio que almacena los archivos de tu programa.

Using absolute paths, you can read files from any location on your system. For now it's easiest to store files in the same directory as your program files, or in a folder such as *text_files* within the directory that stores your program files.

Nota (Note)

Los sistemas Windows utilizan una barra invertida (\) en lugar de una barra inclinada (/) al mostrar las rutas de los archivos, pero debes utilizar barras inclinadas en tu código, incluso en Windows. La biblioteca `pathlib` utilizará automáticamente la representación correcta de la ruta cuando interactúe con tu sistema o con el de cualquier usuario.

Windows systems use a backslash (\) instead of a forward slash (/) when displaying file paths, but you should use forward slashes in your code, even on Windows. The `pathlib` library will automatically use the correct representation of the path when it interacts with your system, or any user's system.

Acceder a las líneas de un archivo (Accessing a File's Lines)

Cuando trabajes con un archivo, a menudo querrás examinar cada línea del mismo. Puede que busques cierta información en el archivo, o puede que quieras modificar el texto del archivo de alguna manera. Por ejemplo, puede que quieras leer un archivo de datos meteorológicos y trabajar con cualquier línea que incluya la palabra *sunny* en la descripción del tiempo de ese día. En un informe de noticias, podrías buscar cualquier línea con la etiqueta `<headline>` y reescribir esa línea con un tipo de formato específico.

When you're working with a file, you'll often want to examine each line of the file. You might be looking for certain information in the file, or you might want to modify the text in the file in some way. For example, you might want to read through a file of weather data and work with any line that includes the word *sunny* in the description of that day's weather. In a news report, you might look for any line with the tag `<headline>` and rewrite that line with a specific kind of formatting.

Puedes utilizar el método `splitlines()` para convertir una cadena larga en un conjunto de líneas, y luego utilizar un bucle `for` para examinar cada línea de un archivo, de una en una:

You can use the `splitlines()` method to turn a long string into a set of lines, and then use a `for` loop to examine each line from a file, one at a time:

lector_archivo.py

file_reader.py

```
from pathlib import Path
```

```
path = Path('pi_digits.txt')
```

```
❶ contents = path.read_text()
```

```
❷ lines = contents.splitlines()
```

```
for line in lines:
```

```
    print(line)
```

Comenzamos leyendo todo el contenido del archivo, como hicimos anteriormente ❶ . Si piensas trabajar con las líneas individuales de un archivo, no necesitas eliminar ningún espacio en blanco al leer el archivo. El método `splitlines()` devuelve una lista de todas las líneas del archivo, y asignamos esta lista a la variable `lines` ❷ . A continuación, hacemos un bucle sobre estas líneas e imprimimos cada una de ellas:

We start out by reading the entire contents of the file, as we did earlier ❶ . If you're planning to work with the individual lines in a file, you don't need to strip any whitespace when reading the file. The `splitlines()` method returns a list of all lines in the file, and we assign this list to the variable `lines` ❷ . We then loop over these lines and print each one:

```
3.1415926535
8979323846
2643383279
```

Como no hemos modificado ninguna de las líneas, la salida coincide exactamente con el archivo de texto original.

Since we haven't modified any of the lines, the output matches the original text file exactly.

Trabajar con el contenido de un archivo *(Working with a File's Contents)*

Después de leer el contenido de un archivo en la memoria, puedes hacer lo que quieras con esos datos, así que vamos a explorar brevemente los dígitos de *pi* . Primero, intentaremos construir una única cadena que contenga todos los dígitos del archivo sin espacios en blanco:

After you've read the contents of a file into memory, you can do whatever you want with that data, so let's briefly explore the digits of *pi* . First, we'll attempt to build a single string containing all the digits in the file with no whitespace in it:

pi_cadena.py

pi_string.py

```
from pathlib import Path
```

```
path = Path('pi_digits.txt')
contents = path.read_text()
```

```
lines = contents.splitlines()
```

```
pi_string = "
```

```
❶ for line in lines:
```

```
    pi_string += line
```

```
print(pi_string)
```

```
print(len(pi_string))
```

Empezamos leyendo el archivo y almacenando cada línea de dígitos en una lista, como hicimos en el ejemplo anterior. A continuación, creamos una variable, `pi_string` , para guardar los dígitos de *pi* . Escribimos un bucle que añade cada línea de dígitos a `pi_string` ❶ . Imprimimos esta cadena y también mostramos su longitud:

We start by reading the file and storing each line of digits in a list, just as we did in the previous example. We then create a variable, `pi_string` , to hold the digits of *pi* . We write a loop that adds each line of digits to `pi_string` ❶ . We print this string, and also show how long the string is:

```
3.1415926535 8979323846 2643383279
36
```

La variable `pi_string` contiene los espacios en blanco que había a la izquierda de los dígitos de cada línea, pero podemos deshacernos de ellos utilizando `rstrip()` en cada línea:

The variable `pi_string` contains the whitespace that was on the left side of the digits in each line, but we can get rid of that by using `rstrip()` on each line:

```
--snip--
```

```
for line in lines:
```

```
    pi_string += line.rstrip()
```

```
print(pi_string)
print(len(pi_string))
```

Ahora tenemos una cadena que contiene *pi* con 30 decimales. La cadena tiene 32 caracteres porque también incluye el 3 inicial y un punto decimal:

Now we have a string containing *pi* to 30 decimal places. The string is 32 characters long because it also includes the leading 3 and a decimal point:

```
3.141592653589793238462643383279
32
```

Nota (Note)

Cuando Python lee de un archivo de texto, interpreta todo el texto del archivo como una cadena. Si lees un número y quieres trabajar con ese valor en un contexto numérico, tendrás que convertirlo en un entero utilizando la función `int()` o en un flotante utilizando la función `float()`.

When Python reads from a text file, it interprets all text in the file as a string. If you read in a number and want to work with that value in a numerical context, you'll have to convert it to an integer using the `int()` function or a float using the `float()` function.

Archivos grandes: Un millón de dígitos (Large Files: One Million Digits)

Hasta ahora, nos hemos centrado en analizar un archivo de texto que sólo contiene tres líneas, pero el código de estos ejemplos funcionaría igual de bien en archivos mucho más grandes. Si empezamos con un archivo de texto que contiene *pi* con 1.000.000 de decimales, en lugar de sólo 30, podemos crear una única cadena que contenga todos estos dígitos. No necesitamos cambiar nuestro programa en absoluto, salvo para pasarle un archivo diferente. También imprimiremos sólo los 50 primeros decimales, para no tener que ver pasar un millón de dígitos en el terminal:

So far, we've focused on analyzing a text file that contains only three lines, but the code in these examples would work just as well on much larger files. If we start with a text file that

contains *pi* to 1,000,000 decimal places, instead of just 30, we can create a single string containing all these digits. We don't need to change our program at all, except to pass it a different file. We'll also print just the first 50 decimal places, so we don't have to watch a million digits scroll by in the terminal:

pi_cadena.py

pi_string.py

```
from pathlib import Path

path = Path('pi_million_digits.txt')
contents = path.read_text()

lines = contents.splitlines()
pi_string = ""
for line in lines:
    pi_string += line.lstrip()

print(f'{pi_string[:52]}...')
print(len(pi_string))
```

La salida muestra que, efectivamente, tenemos una cadena que contiene *pi* con 1.000.000 de decimales:

The output shows that we do indeed have a string containing *pi* to 1,000,000 decimal places:

```
3.14159265358979323846264338327950288419716939937510...
1000002
```

Python no tiene ningún límite inherente a la cantidad de datos con los que puedes trabajar; puedes trabajar con tantos datos como pueda manejar la memoria de tu sistema.

Python has no inherent limit to how much data you can work with; you can work with as much data as your system's memory can handle.

Nota (Note)

Para ejecutar este programa (y muchos de los ejemplos que siguen), necesitarás descargar los recursos disponibles en https://ehmatthes.github.io/pcc_3e.

To run this program (and many of the examples that follow), you'll need to download the resources available at https://ehmatthes.github.io/pcc_3e.

¿Tu cumpleaños está contenido en Pi? (*Is Your Birthday Contained in Pi?*)

Siempre he sentido curiosidad por saber si mi cumpleaños aparece en algún lugar de los dígitos de π . Utilicemos el programa que acabamos de escribir para averiguar si el cumpleaños de alguien aparece en algún lugar del primer millón de dígitos de π . Podemos hacerlo expresando cada cumpleaños como una cadena de dígitos y viendo si esa cadena aparece en algún lugar de `pi_string`:

I've always been curious to know if my birthday appears anywhere in the digits of π . Let's use the program we just wrote to find out if someone's birthday appears anywhere in the first million digits of π . We can do this by expressing each birthday as a string of digits and seeing if that string appears anywhere in `pi_string`:

`pi_cumpleaños.py`

`pi_birthday.py`

```
--snip--
```

```
for line in lines:
```

```
    pi_string += line.strip()
```

```
birthday = input("Enter your birthday, in the form mmddyy: ")
```

```
if birthday in pi_string:
```

```
print("Your birthday appears in the first million digits of pi!")
else:
    print("Your birthday does not appear in the first million digits of pi.")
```

Primero pedimos el cumpleaños del usuario y luego comprobamos si esa cadena está en `pi_string` . Vamos a probarlo:

We first prompt for the user's birthday, and then check if that string is in `pi_string` . Let's try it:

Enter your birthdate, in the form mmddyy: 120372
Your birthday appears in the first million digits of pi!

¡Mi cumpleaños sí aparece en los dígitos de *pi* ! Una vez que hayas leído un archivo, puedes analizar su contenido de cualquier forma que puedas imaginar.

My birthday does appear in the digits of *pi* ! Once you've read from a file, you can analyze its contents in just about any way you can imagine.

Pruébalo tú mismo (*Try It Yourself*)

10-1. Aprender Python: Abre un archivo en blanco en tu editor de texto y escribe unas líneas resumiendo lo que has aprendido sobre Python hasta ahora. Comienza cada línea con la frase *In Python you can.* . . . Guarda el archivo como *learning_python.txt* en el mismo directorio que tus ejercicios de este capítulo. Escribe un programa que lea el archivo e imprima lo que has escrito dos veces: imprime el contenido una vez leyendo todo el archivo, y otra almacenando las líneas en una lista y haciendo un bucle sobre cada línea.

10-1. Learning Python: Open a blank file in your text editor and write a few lines summarizing what you've learned about Python so far. Start each line with the phrase *In Python you can.* . . . Save the file as *learning_python.txt* in the same directory as your exercises from this chapter. Write a program that reads the file and prints what you wrote two times: print the contents once by reading in the entire file, and once by storing the lines in a list and then looping over each line.

10-2. Aprendizaje de C: Puedes utilizar el método `replace()` para sustituir cualquier palabra de una cadena por otra diferente. Aquí tienes un ejemplo rápido que muestra cómo sustituir 'dog' por 'cat' en una frase:

10-2. Learning C: You can use the `replace()` method to replace any word in a string with a different word. Here's a quick example showing how to replace 'dog' with 'cat' in a sentence:

```
>>> message = "I really like dogs."
>>> message.replace('dog', 'cat')
'I really like cats.'
```

Lee cada línea del archivo que acabas de crear, *learning_python.txt*, y sustituye la palabra *Python* por el nombre de otro idioma, como *C*. Imprime cada línea modificada en la pantalla.

Read in each line from the file you just created, *learning_python.txt*, and replace the word *Python* with the name of another language, such as *C*. Print each modified line to the screen.

10-3. Código más sencillo : El programa *file_reader.py* de esta sección utiliza una variable temporal, `lines`, para mostrar cómo funciona `splitlines()`. Puedes omitir la variable temporal y hacer un bucle directamente sobre la lista que devuelve `splitlines()` :

10-3. Simpler Code : The program *file_reader.py* in this section uses a temporary variable, `lines`, to show how `splitlines()` works. You can skip the temporary variable and loop directly over the list that `splitlines()` returns:

for line in contents.splitlines():

Elimina la variable temporal de cada uno de los programas de esta sección, para hacerlos más concisos.

Remove the temporary variable from each of the programs in this section, to make them more concise.

Escribir en un archivo (*Writing to a File*)

Una de las formas más sencillas de guardar datos es escribirlos en un archivo. Cuando escribes texto en un archivo, la salida seguirá estando disponible después de que cierres el terminal que contiene la salida de tu programa. Puedes examinar la salida después de que un programa termine de ejecutarse, y también puedes compartir los archivos de salida con otras personas. También puedes escribir programas que vuelvan a leer el texto en la memoria y trabajar con él más tarde.

One of the simplest ways to save data is to write it to a file. When you write text to a file, the output will still be available after you close the terminal containing your program's output. You can examine output after a program finishes running, and you can share the output files with others as well. You can also write programs that read the text back into memory and work with it again later.

Escribir una sola línea (*Writing a Single Line*)

Una vez que tengas definida una ruta, puedes escribir en un archivo utilizando el método `write_text()` . Para ver cómo funciona, escribamos un mensaje sencillo y almacenémoslo en un archivo en lugar de imprimirlo en la pantalla:

Once you have a path defined, you can write to a file using the `write_text()` method. To see how this works, let's write a simple message and store it in a file instead of printing it to the screen:

escribir_mensaje.py

write_message.py

```
from pathlib import Path
```

```
path = Path('programming.txt')  
path.write_text("I love programming.")
```

El método `write_text()` toma un único argumento: la cadena que quieres escribir en el archivo. Este programa no tiene salida de terminal, pero si abres el archivo *programming.txt* , verás una línea:

The `write_text()` method takes a single argument: the string that you want to write to the file. This program has no terminal output, but if you open the file *programming.txt* , you'll see one line:

programacion.txt

programming.txt

I love programming.

Este archivo se comporta como cualquier otro archivo de tu ordenador. Puedes abrirlo, escribir nuevo texto en él, copiar de él, pegar en él, etc.

This file behaves like any other file on your computer. You can open it, write new text in it, copy from it, paste to it, and so forth.

Nota (Note)

Python sólo puede escribir cadenas en un archivo de texto. Si quieres almacenar datos numéricos en un archivo de texto, tendrás que convertirlos primero a formato de cadena mediante la función `str()` .

Python can only write strings to a text file. If you want to store numerical data in a text file, you'll have to convert the data to string format first using the `str()` function.

Escribir varias líneas (*Writing Multiple Lines*)

El método `write_text()` hace algunas cosas entre bastidores. Si el archivo al que apunta `path` no existe, lo crea. Además, después de escribir la cadena en el archivo, se asegura de que el archivo se cierra correctamente. Los archivos que no se cierran correctamente pueden hacer que falten datos o que se corrompan.

The `write_text()` method does a few things behind the scenes. If the file that `path` points to doesn't exist, it creates that file. Also, after writing the string to the file, it makes sure the file is closed properly. Files that aren't closed properly can lead to missing or corrupted data.

Para escribir más de una línea en un archivo, tienes que construir una cadena que contenga todo el contenido del archivo, y luego llamar a `write_text()` con esa cadena. Vamos a escribir varias líneas en el archivo *programming.txt* :

To write more than one line to a file, you need to build a string containing the entire contents of the file, and then call `write_text()` with that string. Let's write several lines to the *programming.txt* file:

```
from pathlib import Path
```

```
contents = "I love programming.\n"  
contents += "I love creating new games.\n"  
contents += "I also love working with data.\n"
```

```
path = Path('programming.txt')  
path.write_text(contents)
```

Definimos una variable llamada `contents` que contendrá todo el contenido del archivo. En la siguiente línea, utilizamos el operador `+=` para añadir a esta cadena. Puedes hacer esto tantas veces como necesites, para construir cadenas de cualquier longitud. En este caso incluimos caracteres de nueva línea al final de cada línea, para asegurarnos de que cada declaración aparece en su propia línea.

We define a variable called `contents` that will hold the entire contents of the file. On the next line, we use the `+=` operator to add to this string. You can do this as many times as

you need, to build strings of any length. In this case we include newline characters at the end of each line, to make sure each statement appears on its own line.

Si ejecutas esto y luego abres *programming.txt* , verás cada una de estas líneas en el archivo de texto:

If you run this and then open *programming.txt* , you'll see each of these lines in the text file:

I love programming.
I love creating new games.
I also love working with data.

También puedes utilizar espacios, caracteres de tabulación y líneas en blanco para dar formato a tu salida, igual que has estado haciendo con la salida basada en terminal. No hay límite para la longitud de tus cadenas, y así es como se crean muchos documentos generados por ordenador.

You can also use spaces, tab characters, and blank lines to format your output, just as you've been doing with terminal-based output. There's no limit to the length of your strings, and this is how many computer-generated documents are created.

Nota (Note)

Ten cuidado al llamar a `write_text()` sobre un objeto ruta. Si el archivo ya existe, `write_text()` borrará el contenido actual del archivo y escribirá nuevo contenido en él. Más adelante en este capítulo, aprenderás a comprobar si un archivo existe utilizando `pathlib` .

Be careful when calling `write_text()` on a path object. If the file already exists, `write_text()` will erase the current contents of the file and write new contents to the file. Later in this chapter, you'll learn to check whether a file exists using `pathlib` .

Pruébalo tú mismo (Try It Yourself)

10-4. Invitado: Escribe un programa que pregunte al usuario su nombre. Cuando responda, escribe su nombre en un archivo llamado *guest.txt*.

10-4. Guest: Write a program that prompts the user for their name. When they respond, write their name to a file called *guest.txt*.

10-5. Libro de visitas: Escribe un bucle `while` que pregunte a los usuarios por su nombre. Recoge todos los nombres introducidos y escríbelos en un archivo llamado *guest_book.txt*. Asegúrate de que cada entrada aparece en una nueva línea del archivo.

10-5. Guest Book: Write a `while` loop that prompts users for their name. Collect all the names that are entered, and then write these names to a file called *guest_book.txt*. Make sure each entry appears on a new line in the file.

Excepciones (Exceptions)

Python utiliza objetos especiales llamados *exceptions* para gestionar los errores que surgen durante la ejecución de un programa. Cada vez que se produce un error que hace que Python no sepa qué hacer a continuación, crea un objeto de excepción. Si escribes código que maneje la excepción, el programa seguirá ejecutándose. Si no manejas la excepción, el programa se detendrá y mostrará un mensaje *traceback*, que incluye un informe de la excepción que se ha producido.

Python uses special objects called *exceptions* to manage errors that arise during a program's execution. Whenever an error occurs that makes Python unsure of what to do next, it creates an exception object. If you write code that handles the exception, the program will continue running. If you don't handle the exception, the program will halt and show a *traceback*, which includes a report of the exception that was raised.

Las excepciones se manejan con los bloques `try - except`. Un bloque `try - except` pide a Python que haga algo, pero también le dice qué hacer si se produce una excepción. Si utilizas los bloques `try - except`, tus programas seguirán ejecutándose aunque las cosas empiecen a ir mal. En lugar de rastreos, que pueden resultar confusos de leer para los

usuarios, éstos verán mensajes de error amigables que tú has escrito.

Exceptions are handled with `try - except` blocks. A *try - except* block asks Python to do something, but it also tells Python what to do if an exception is raised. When you use `try - except` blocks, your programs will continue running even if things start to go wrong. Instead of tracebacks, which can be confusing for users to read, users will see friendly error messages that you've written.

Manejo de la excepción `ZeroDivisionError` (*Handling the `ZeroDivisionError` Exception*)

Veamos un error sencillo que hace que Python lance una excepción. Probablemente sepas que es imposible dividir un número por cero, pero vamos a pedirle a Python que lo haga de todos modos:

Let's look at a simple error that causes Python to raise an exception. You probably know that it's impossible to divide a number by zero, but let's ask Python to do it anyway:

calculadora_division.py

division_calculator.py

```
print(5/0)
```

Python no puede hacerlo, por lo que obtenemos un rastreo:

Python can't do this, so we get a traceback:

```
Traceback (most recent call last):
  File "division_calculator.py", line 1, in <module>
    print(5/0)
    ~^~
```

❶ `ZeroDivisionError: division by zero`

El error del que se informa en el rastreo, `ZeroDivisionError` , es un objeto de excepción ❶ . Python crea este tipo de objeto en respuesta a una situación en la que no puede hacer lo que le pedimos. Cuando esto ocurre, Python detiene el programa y nos indica el tipo de excepción que se ha producido. Podemos utilizar esta información para modificar nuestro programa. Le diremos a Python qué hacer cuando se produzca este tipo de excepción; así, si vuelve a ocurrir, estaremos preparados.

The error reported in the traceback, `ZeroDivisionError` , is an exception object ❶ . Python creates this kind of object in response to a situation where it can't do what we ask it to. When this happens, Python stops the program and tells us the kind of exception that was raised. We can use this information to modify our program. We'll tell Python what to do when this kind of exception occurs; that way, if it happens again, we'll be prepared.

Utilizar bloques try-except *(Using try-except Blocks)*

Cuando creas que se puede producir un error, puedes escribir un bloque `try - except` para manejar la excepción que se pueda producir. Le dices a Python que intente ejecutar algún código, y le dices qué hacer si el código da lugar a un tipo concreto de excepción.

When you think an error may occur, you can write a `try - except` block to handle the exception that might be raised. You tell Python to try running some code, and you tell it what to do if the code results in a particular kind of exception.

Este es el aspecto de un bloque `try - except` para manejar la excepción `ZeroDivisionError` :

Here's what a `try - except` block for handling the `ZeroDivisionError` exception looks like:

```
try:
    print(5/0)
except ZeroDivisionError:
    print("You can't divide by zero!")
```

Colocamos `print(5/0)` , la línea que causó el error, dentro de un bloque `try` . Si el código del bloque `try` funciona, Python se salta el bloque `except` . Si el código del bloque `try`

provoca un error, Python busca un bloque `except` cuyo error coincida con el que se ha producido, y ejecuta el código de ese bloque.

We put `print(5/0)` , the line that caused the error, inside a `try` block. If the code in a `try` block works, Python skips over the `except` block. If the code in the `try` block causes an error, Python looks for an `except` block whose error matches the one that was raised, and runs the code in that block.

En este ejemplo, el código del bloque `try` produce un `ZeroDivisionError` , por lo que Python busca un bloque `except` que le indique cómo responder. Python ejecuta entonces el código de ese bloque, y el usuario ve un mensaje de error amigable en lugar de un rastreo:

In this example, the code in the `try` block produces a `ZeroDivisionError` , so Python looks for an `except` block telling it how to respond. Python then runs the code in that block, and the user sees a friendly error message instead of a traceback:

You can't divide by zero!

Si hubiera más código después del bloque `try - except` , el programa seguiría ejecutándose porque le hemos dicho a Python cómo manejar el error. Veamos un ejemplo en el que la captura de un error puede permitir que un programa siga ejecutándose.

If more code followed the `try - except` block, the program would continue running because we told Python how to handle the error. Let's look at an example where catching an error can allow a program to continue running.

Utilizar excepciones para evitar fallos (*Using Exceptions to Prevent Crashes*)

Manejar los errores correctamente es especialmente importante cuando el programa tiene más trabajo que hacer después de que se produzca el error. Esto ocurre a menudo en los programas que piden a los usuarios que introduzcan datos. Si el programa responde adecuadamente a las entradas no válidas, puede pedir más entradas válidas en lugar de bloquearse.

Handling errors correctly is especially important when the program has more work to do after the error occurs. This happens often in programs that prompt users for input. If the program responds to invalid input appropriately, it can prompt for more valid input instead of crashing.

Creemos una calculadora sencilla que sólo haga divisiones:

Let's create a simple calculator that does only division:

calculadora_division.py

division_calculator.py

```
print("Give me two numbers, and I'll divide them.")
print("Enter 'q' to quit.")

while True:
    ❶ first_number = input("\nFirst number: ")
    if first_number == 'q':
        break
    ❷ second_number = input("Second number: ")
    if second_number == 'q':
        break
    ❸ answer = int(first_number) / int(second_number)
    print(answer)
```

Este programa pide al usuario que introduzca un `first_number` ❶ y, si no introduce `q` para salir, un `second_number` ❷. A continuación, dividimos estos dos números para obtener un `answer` ❸. Este programa no hace nada para manejar los errores, por lo que pedirle que divida por cero hace que se bloquee:

This program prompts the user to input a `first_number` ❶ and, if the user does not enter `q` to quit, a `second_number` ❷. We then divide these two numbers to get an `answer` ❸. This program does nothing to handle errors, so asking it to divide by zero causes it to crash:

Give me two numbers, and I'll divide them.
Enter 'q' to quit.

First number: 5

Second number: 0

Traceback (most recent call last):

```
File "division_calculator.py", line 11, in <module>
    answer = int(first_number) / int(second_number)
              ~~~~~^~~~~~
```

ZeroDivisionError: division by zero

Es malo que el programa se bloquee, pero tampoco es buena idea dejar que los usuarios vean las trazas. Los usuarios no técnicos se sentirán confundidos por ellos, y en un entorno malicioso, los atacantes aprenderán más de lo que tú quieres. Por ejemplo, sabrán el nombre del archivo de tu programa y verán una parte de tu código que no funciona correctamente. A veces, un atacante hábil puede utilizar esta información para determinar qué tipo de ataques utilizar contra tu código.

It's bad that the program crashed, but it's also not a good idea to let users see tracebacks. Nontechnical users will be confused by them, and in a malicious setting, attackers will learn more than you want them to. For example, they'll know the name of your program file, and they'll see a part of your code that isn't working properly. A skilled attacker can sometimes use this information to determine which kind of attacks to use against your code.

El bloque else (*The else Block*)

Podemos hacer que este programa sea más resistente a los errores envolviendo la línea que podría producir errores en un bloque `try - except`. El error se produce en la línea que realiza la división, así que ahí es donde pondremos el bloque `try - except`. Este ejemplo también incluye un bloque `else`. Cualquier código que dependa de que el bloque `try` se ejecute correctamente va en el bloque `else`:

We can make this program more error resistant by wrapping the line that might produce errors in a `try - except` block. The error occurs on the line that performs the division, so that's where we'll put the `try - except` block. This example also includes an `else` block. Any code that depends on the `try` block executing successfully goes in the `else` block:

```

--snip--
while True:
    --snip--
    if second_number == 'q':
        break
    ❶ try:
        answer = int(first_number) / int(second_number)
    ❷ except ZeroDivisionError:
        print("You can't divide by 0!")
    ❸ else:
        print(answer)

```

Pedimos a Python que intente completar la operación de división en un bloque `try` ❶ , que incluye sólo el código que podría causar un error. Cualquier código que dependa del éxito del bloque `try` se añade al bloque `else` . En este caso, si la operación de división tiene éxito, utilizamos el bloque `else` para imprimir el resultado ❸ .

We ask Python to try to complete the division operation in a `try` block ❶ , which includes only the code that might cause an error. Any code that depends on the `try` block succeeding is added to the `else` block. In this case, if the division operation is successful, we use the `else` block to print the result ❸ .

El bloque `except` indica a Python cómo responder cuando surge un `ZeroDivisionError` ❷ . Si el bloque `try` no tiene éxito debido a un error de división por cero, imprimimos un mensaje amigable indicando al usuario cómo evitar este tipo de error. El programa sigue ejecutándose y el usuario nunca ve un rastreo:

The `except` block tells Python how to respond when a `ZeroDivisionError` arises ❷ . If the `try` block doesn't succeed because of a division-by-zero error, we print a friendly message telling the user how to avoid this kind of error. The program continues to run, and the user never sees a traceback:

Give me two numbers, and I'll divide them.
Enter 'q' to quit.

```

First number: 5
Second number: 0
You can't divide by 0!

```

First number: 5
Second number: 2
2.5

First number: q

El único código que debe ir en un bloque `try` es el código que pueda provocar el lanzamiento de una excepción. A veces tendrás código adicional que sólo debe ejecutarse si el bloque `try` ha tenido éxito; este código va en el bloque `else` . El bloque `except` indica a Python qué hacer en caso de que se produzca una determinada excepción al intentar ejecutar el código del bloque `try` .

The only code that should go in a `try` block is code that might cause an exception to be raised. Sometimes you'll have additional code that should run only if the `try` block was successful; this code goes in the `else` block. The `except` block tells Python what to do in case a certain exception arises when it tries to run the code in the `try` block.

Al anticiparte a las posibles fuentes de error, puedes escribir programas robustos que sigan ejecutándose aunque se encuentren con datos no válidos y recursos ausentes. Tu código será resistente a los errores inocentes del usuario y a los ataques maliciosos.

By anticipating likely sources of errors, you can write robust programs that continue to run even when they encounter invalid data and missing resources. Your code will be resistant to innocent user mistakes and malicious attacks.

Manejo de la Excepción `FileNotFoundError` (*Handling the `FileNotFoundError` Exception*)

Un problema común cuando se trabaja con archivos es la gestión de archivos perdidos. El archivo que buscas puede estar en una ubicación diferente, el nombre del archivo puede estar mal escrito o puede no existir en absoluto. Puedes manejar todas estas situaciones con un bloque `try - except` .

One common issue when working with files is handling missing files. The file you're looking for might be in a different location, the filename might be misspelled, or the file might not exist at all. You can handle all of these situations with a `try - except` block.

Intentemos leer un archivo que no existe. El siguiente programa intenta leer el contenido de *Alice in Wonderland* , pero no he guardado el archivo *alice.txt* en el mismo directorio que *alice.py* :

Let's try to read a file that doesn't exist. The following program tries to read in the contents of *Alice in Wonderland* , but I haven't saved the file *alice.txt* in the same directory as *alice.py* :

alice.py

alice.py

```
from pathlib import Path
```

```
path = Path('alice.txt')
contents = path.read_text(encoding='utf-8')
```

Ten en cuenta que aquí estamos utilizando `read_text()` de una forma ligeramente distinta a la que has visto antes. El argumento `encoding` es necesario cuando la codificación por defecto de tu sistema no coincide con la codificación del archivo que se está leyendo. Lo más probable es que esto ocurra al leer de un archivo que no se creó en tu sistema.

Note that we're using `read_text()` in a slightly different way here than what you saw earlier. The `encoding` argument is needed when your system's default encoding doesn't match the encoding of the file that's being read. This is most likely to happen when reading from a file that wasn't created on your system.

Python no puede leer de un archivo que falta, por lo que lanza una excepción:

Python can't read from a missing file, so it raises an exception:

Traceback (most recent call last):

❶ File "alice.py", line 4, in <module>

Éste es un rastreo más largo que los que hemos visto anteriormente, así que veamos cómo dar sentido a rastreos más complejos. A menudo es mejor empezar por el final del rastreo. En la última línea, podemos ver que se lanzó una excepción `FileNotFoundError` ③. Esto es importante porque nos indica qué tipo de excepción utilizar en el bloque `except` que vamos a escribir.

Si miramos hacia atrás, cerca del principio del rastreo ❶, podemos ver que el error se produjo en la línea 4 del archivo *alice.py*. La siguiente línea muestra la línea de código que causó el error ❷. El resto del rastreo muestra código de las bibliotecas que intervienen en la apertura y lectura de archivos. Normalmente no es necesario que leas o comprendas todas estas líneas de un rastreo.

Para tratar el error que se está produciendo, el bloque `try` comenzará con la línea que se identificó como problemática en el rastreo. En nuestro ejemplo, es la línea que contiene `read_text()` :

To handle the error that's being raised, the `try` block will begin with the line that was identified as problematic in the traceback. In our example, this is the line that contains `read_text()` :

```
from pathlib import Path

path = Path('alice.txt')
try:
    contents = path.read_text(encoding='utf-8')
❶ except FileNotFoundError:
    print(f"Sorry, the file {path} does not exist.")
```

En este ejemplo, el código del bloque `try` produce un `FileNotFoundError` , así que escribimos un bloque `except` que coincide con ese error ❶ . Python ejecuta entonces el código de ese bloque cuando no se encuentra el archivo, y el resultado es un mensaje de error amigable en lugar de un rastreo:

In this example, the code in the `try` block produces a `FileNotFoundError` , so we write an `except` block that matches that error ❶ . Python then runs the code in that block when the file can't be found, and the result is a friendly error message instead of a traceback:

Sorry, the file `alice.txt` does not exist.

El programa no tiene nada más que hacer si el archivo no existe, así que ésta es toda la salida que vemos. Continuemos con este ejemplo y veamos cómo puede ayudar el manejo de excepciones cuando trabajas con más de un archivo.

The program has nothing more to do if the file doesn't exist, so this is all the output we see. Let's build on this example and see how exception handling can help when you're working with more than one file.

Analizar texto (*Analyzing Text*)

Puedes analizar archivos de texto que contengan libros enteros. Muchas obras clásicas de la literatura están disponibles como simples archivos de texto porque son de dominio público. Los textos utilizados en esta sección proceden del Proyecto Gutenberg (<https://gutenberg.org>). El Proyecto Gutenberg mantiene una colección de obras literarias que están disponibles en el dominio público, y es un gran recurso si te interesa trabajar con

textos literarios en tus proyectos de programación.

You can analyze text files containing entire books. Many classic works of literature are available as simple text files because they are in the public domain. The texts used in this section come from Project Gutenberg (<https://gutenberg.org>). Project Gutenberg maintains a collection of literary works that are available in the public domain, and it's a great resource if you're interested in working with literary texts in your programming projects.

Extraigamos el texto de *Alice in Wonderland* e intentemos contar el número de palabras del texto. Para ello, utilizaremos el método de cadena `split()` , que por defecto divide una cadena siempre que encuentra algún espacio en blanco:

Let's pull in the text of *Alice in Wonderland* and try to count the number of words in the text. To do this, we'll use the string method `split()` , which by default splits a string wherever it finds any whitespace:

```
from pathlib import Path

path = Path('alice.txt')
try:
    contents = path.read_text(encoding='utf-8')
except FileNotFoundError:
    print(f'Sorry, the file {path} does not exist.')
else:
    # Count the approximate number of words in the file:
    ❶ words = contents.split()
    ❷ num_words = len(words)
    print(f'The file {path} has about {num_words} words.')
```

He movido el archivo *alice.txt* al directorio correcto, así que el bloque `try` funcionará esta vez. Tomamos la cadena `contents` , que ahora contiene todo el texto de *Alice in Wonderland* como una cadena larga, y utilizamos `split()` para producir una lista de todas las palabras del libro ❶ . Utilizando `len()` en esta lista ❷ obtenemos una buena aproximación del número de palabras del texto original. Por último, imprimimos una sentencia que informa de cuántas palabras se han encontrado en el archivo. Este código se coloca en el bloque `else` porque sólo funciona si el código del bloque `try` se ejecutó correctamente.

I moved the file *alice.txt* to the correct directory, so the `try` block will work this time. We take the string `contents`, which now contains the entire text of *Alice in Wonderland* as one long string, and use `split()` to produce a list of all the words in the book ❶. Using `len()` on this list ❷ gives us a good approximation of the number of words in the original text. Lastly, we print a statement that reports how many words were found in the file. This code is placed in the `else` block because it only works if the code in the `try` block was executed successfully.

La salida nos dice cuántas palabras hay en *alice.txt* :

The output tells us how many words are in *alice.txt* :

The file *alice.txt* has about 29594 words.

El recuento es un poco alto porque el editor proporciona información adicional en el archivo de texto utilizado aquí, pero es una buena aproximación a la longitud de *Alice in Wonderland*.

The count is a little high because extra information is provided by the publisher in the text file used here, but it's a good approximation of the length of *Alice in Wonderland*.

Trabajar con varios archivos (*Working with Multiple Files*)

Añadamos más libros para analizar, pero antes traslademos el grueso de este programa a una función llamada `count_words()`. Esto facilitará la ejecución del análisis para varios libros:

Let's add more books to analyze, but before we do, let's move the bulk of this program to a function called `count_words()`. This will make it easier to run the analysis for multiple books:

word_count.py

word_count.py

```
from pathlib import Path

def count_words(path):
    ❶ """Count the approximate number of words in a file."""
    try:
        contents = path.read_text(encoding='utf-8')
    except FileNotFoundError:
        print(f"Sorry, the file {path} does not exist.")
    else:
        # Count the approximate number of words in the file:
        words = contents.split()
        num_words = len(words)
        print(f"The file {path} has about {num_words} words.")

path = Path('alice.txt')
count_words(path)
```

La mayor parte de este código no ha cambiado. Sólo se le ha aplicado una sangría y se ha trasladado al cuerpo de `count_words()` . Es una buena costumbre mantener actualizados los comentarios cuando modificas un programa, así que el comentario también se ha cambiado por un docstring y se ha reformulado ligeramente ❶ .

Most of this code is unchanged. It's only been indented, and moved into the body of `count_words()` . It's a good habit to keep comments up to date when you're modifying a program, so the comment has also been changed to a docstring and reworded slightly ❶ .

Ahora podemos escribir un bucle corto para contar las palabras de cualquier texto que queramos analizar. Para ello, almacenamos los nombres de los archivos que queremos analizar en una lista y, a continuación, llamamos a `count_words()` para cada archivo de la lista. Intentaremos contar las palabras de *Alice in Wonderland* , *Siddhartha* , *Moby Dick* , y *Little Women* , que son de dominio público. He dejado intencionadamente *siddhartha.txt* fuera del directorio que contiene *word_count.py* , para que podamos ver lo bien que nuestro programa gestiona un archivo que falta:

Now we can write a short loop to count the words in any text we want to analyze. We do this by storing the names of the files we want to analyze in a list, and then we call `count_words()` for each file in the list. We'll try to count the words for *Alice in Wonderland* , *Siddhartha* , *Moby Dick* , and *Little Women* , which are all available in the public domain. I've intentionally left *siddhartha.txt* out of the directory containing *word_count.py* , so we can see how well our program handles a missing file:

```
from pathlib import Path

def count_words(filename):
    --snip--

filenames = ['alice.txt', 'siddhartha.txt', 'moby_dick.txt',
             'little_women.txt']
for filename in filenames:
    ❶ path = Path(filename)
    count_words(path)
```

Los nombres de los archivos se almacenan como simples cadenas. A continuación, cada cadena se convierte en un objeto `Path` ❶ , antes de la llamada a `count_words()` . El archivo *siddhartha.txt* que falta no tiene ningún efecto sobre el resto de la ejecución del programa:

The names of the files are stored as simple strings. Each string is then converted to a `Path` object ❶ , before the call to `count_words()` . The missing *siddhartha.txt* file has no effect on the rest of the program's execution:

The file *alice.txt* has about 29594 words.
Sorry, the file *siddhartha.txt* does not exist.
The file *moby_dick.txt* has about 215864 words.
The file *little_women.txt* has about 189142 words.

Utilizar el bloque `try - except` en este ejemplo proporciona dos ventajas significativas. Evitamos que nuestros usuarios vean un rastreo, y dejamos que el programa siga analizando los textos que es capaz de encontrar. Si no atrapamos el `FileNotFoundError` que plantea *siddhartha.txt* , el usuario vería un rastreo completo, y el programa dejaría de ejecutarse tras intentar analizar *Siddhartha* . Nunca analizaría *Moby Dick* ni *Little Women* .

Using the `try - except` block in this example provides two significant advantages. We prevent our users from seeing a traceback, and we let the program continue analyzing the texts it's able to find. If we don't catch the `FileNotFoundError` that *siddhartha.txt* raises, the user would see a full traceback, and the program would stop running after trying to analyze *Siddhartha*. It would never analyze *Moby Dick* or *Little Women*.

Fallar en silencio (*Failing Silently*)

En el ejemplo anterior, informamos a nuestros usuarios de que uno de los archivos no estaba disponible. Pero no es necesario que informes de todas las excepciones que detectes. A veces, querrás que el programa falle silenciosamente cuando se produzca una excepción y continúe como si no hubiera pasado nada. Para hacer que un programa falle silenciosamente, escribe un bloque `try` como de costumbre, pero dile explícitamente a Python que no haga nada en el bloque `except`. Python tiene una sentencia `pass` que le dice que no haga nada en un bloque:

In the previous example, we informed our users that one of the files was unavailable. But you don't need to report every exception you catch. Sometimes, you'll want the program to fail silently when an exception occurs and continue on as if nothing happened. To make a program fail silently, you write a `try` block as usual, but you explicitly tell Python to do nothing in the `except` block. Python has a `pass` statement that tells it to do nothing in a block:

```
def count_words(path):
    """Count the approximate number of words in a file."""
    try:
        --snip--
    except FileNotFoundError:
        pass
    else:
        --snip--
```

La única diferencia entre este listado y el anterior es la sentencia `pass` en el bloque `except`. Ahora, cuando se lanza un `FileNotFoundError`, el código del bloque `except` se ejecuta, pero no ocurre nada. No se produce ningún rastreo, y no hay ninguna salida en respuesta al error que se ha producido. Los usuarios ven el recuento de palabras de cada archivo que existe, pero no ven ninguna indicación de que no se ha encontrado un archivo:

The only difference between this listing and the previous one is the `pass` statement in the `except` block. Now when a `FileNotFoundError` is raised, the code in the `except` block runs, but nothing happens. No traceback is produced, and there's no output in response to the error that was raised. Users see the word counts for each file that exists, but they don't see any indication that a file wasn't found:

The file `alice.txt` has about 29594 words.
The file `moby_dick.txt` has about 215864 words.
The file `little_women.txt` has about 189142 words.

La declaración `pass` también actúa como marcador de posición. Es un recordatorio de que estás eligiendo no hacer nada en un punto concreto de la ejecución de tu programa y que puede que quieras hacer algo allí más adelante. Por ejemplo, en este programa podríamos decidir escribir cualquier nombre de archivo que falte en un archivo llamado *missing_files.txt*. Nuestros usuarios no verían este archivo, pero nosotros podríamos leerlo y ocuparnos de cualquier texto que faltara.

The `pass` statement also acts as a placeholder. It's a reminder that you're choosing to do nothing at a specific point in your program's execution and that you might want to do something there later. For example, in this program we might decide to write any missing filenames to a file called *missing_files.txt*. Our users wouldn't see this file, but we'd be able to read the file and deal with any missing texts.

Decidir de qué errores informar (*Deciding Which Errors to Report*)

¿Cómo sabes cuándo informar de un error a tus usuarios y cuándo dejar que tu programa falle silenciosamente? Si los usuarios saben qué textos se supone que deben analizarse, es posible que aprecien un mensaje que les informe de por qué no se analizaron algunos textos. Si los usuarios esperan ver algunos resultados, pero no saben qué libros se supone que deben analizarse, puede que no necesiten saber que algunos textos no estaban disponibles. Dar a los usuarios información que no están buscando puede disminuir la usabilidad de tu programa. Las estructuras de gestión de errores de Python te dan un control preciso sobre cuánto compartir con los usuarios cuando las cosas van mal; tú decides cuánta información compartir.

How do you know when to report an error to your users and when to let your program fail silently? If users know which texts are supposed to be analyzed, they might appreciate a message informing them why some texts were not analyzed. If users expect to see some results but don't know which books are supposed to be analyzed, they might not need to know that some texts were unavailable. Giving users information they aren't looking for

can decrease the usability of your program. Python's error-handling structures give you fine-grained control over how much to share with users when things go wrong; it's up to you to decide how much information to share.

El código bien escrito y correctamente probado no es muy propenso a errores internos, como errores sintácticos o lógicos. Pero cada vez que tu programa depende de algo externo, como la entrada de datos del usuario, la existencia de un archivo o la disponibilidad de una conexión de red, existe la posibilidad de que se produzca una excepción. Un poco de experiencia te ayudará a saber dónde incluir bloques de gestión de excepciones en tu programa y cuánto informar a los usuarios sobre los errores que surjan.

Well-written, properly tested code is not very prone to internal errors, such as syntax or logical errors. But every time your program depends on something external such as user input, the existence of a file, or the availability of a network connection, there is a possibility of an exception being raised. A little experience will help you know where to include exception-handling blocks in your program and how much to report to users about errors that arise.

Pruébalo tú mismo (*Try It Yourself*)

10-6. Suma: Un problema común cuando se solicita una entrada numérica ocurre cuando la gente proporciona texto en lugar de números. Cuando intentes convertir la entrada en un `int`, obtendrás un `ValueError`. Escribe un programa que pida dos números. Súmalos e imprime el resultado. Atrapa el `ValueError` si alguno de los valores de entrada no es un número, e imprime un mensaje de error amigable. Prueba tu programa introduciendo dos números y luego introduciendo un texto en lugar de un número.

10-6. Addition: One common problem when prompting for numerical input occurs when people provide text instead of numbers. When you try to convert the input to an `int`, you'll get a `ValueError`. Write a program that prompts for two numbers. Add them together and print the result. Catch the `ValueError` if either input value is not a number, and print a friendly error message. Test your program by entering two numbers and then by entering some text instead of a number.

10-7. Calculadora de sumas: Envuelve tu código del Ejercicio 10-5 en un bucle `while` para que el usuario pueda seguir introduciendo números, aunque se equivoque e introduzca

texto en lugar de un número.

10-7. Addition Calculator: Wrap your code from Exercise 10-5 in a `while` loop so the user can continue entering numbers, even if they make a mistake and enter text instead of a number.

10-8. Gatos y Perros: Crea dos archivos, *cats.txt* y *dogs.txt* . Guarda al menos tres nombres de gatos en el primer fichero y tres nombres de perros en el segundo. Escribe un programa que intente leer estos archivos e imprima su contenido en la pantalla. Envuelve tu código en un bloque `try-except` para detectar el error `FileNotFound` , e imprime un mensaje amigable si falta un archivo. Mueve uno de los archivos a una ubicación diferente de tu sistema, y asegúrate de que el código del bloque `except` se ejecuta correctamente.

10-8. Cats and Dogs: Make two files, *cats.txt* and *dogs.txt* . Store at least three names of cats in the first file and three names of dogs in the second file. Write a program that tries to read these files and print the contents of the file to the screen. Wrap your code in a `try-except` block to catch the `FileNotFound` error, and print a friendly message if a file is missing. Move one of the files to a different location on your system, and make sure the code in the `except` block executes properly.

10-9. Perros y gatos silenciosos: Modifica tu bloque `except` del Ejercicio 10-7 para que falle silenciosamente si falta alguno de los archivos.

10-9. Silent Cats and Dogs: Modify your `except` block in Exercise 10-7 to fail silently if either file is missing.

10-10. Palabras comunes: Visita el Proyecto Gutenberg (<https://gutenberg.org>) y encuentra algunos textos que te gustaría analizar. Descarga los archivos de texto de estas obras, o copia el texto en bruto desde tu navegador a un archivo de texto en tu ordenador.

10-10. Common Words: Visit Project Gutenberg (<https://gutenberg.org>) and find a few texts you'd like to analyze. Download the text files for these works, or copy the raw text from your browser into a text file on your computer.

Puedes utilizar el método `count()` para averiguar cuántas veces aparece una palabra o frase en una cadena. Por ejemplo, el siguiente código cuenta el número de veces que aparece 'row' en una cadena:

You can use the `count()` method to find out how many times a word or phrase appears in a string. For example, the following code counts the number of times 'row' appears in a string:

```
>>> line = "Row, row, row your boat"
>>> line.count('row')
2
>>> line.lower().count('row')
3
```

Observa que la conversión de la cadena a minúsculas mediante `lower()` capta todas las apariciones de la palabra que buscas, independientemente de cómo esté formateada.

Notice that converting the string to lowercase using `lower()` catches all appearances of the word you're looking for, regardless of how it's formatted.

Escribe un programa que lea los archivos que has encontrado en el Proyecto Gutenberg y determine cuántas veces aparece la palabra 'the' en cada texto. Esto será una aproximación porque también contará palabras como 'then' y 'there' . Prueba a contar 'the ' , con un espacio en la cadena, y verás cuánto más baja es tu cuenta.

Write a program that reads the files you found at Project Gutenberg and determines how many times the word 'the' appears in each text. This will be an approximation because it will also count words such as 'then' and 'there' . Try counting 'the ' , with a space in the string, and see how much lower your count is.

Almacenamiento de datos (*Storing Data*)

Muchos de tus programas pedirán a los usuarios que introduzcan cierto tipo de información. Podrías permitir a los usuarios almacenar preferencias en un juego o

proporcionar datos para una visualización. Sea cual sea el enfoque de tu programa, almacenarás la información que los usuarios proporcionen en estructuras de datos como listas y diccionarios. Cuando los usuarios cierren un programa, casi siempre querrás guardar la información que introdujeron. Una forma sencilla de hacerlo consiste en almacenar tus datos utilizando el módulo `json` .

Many of your programs will ask users to input certain kinds of information. You might allow users to store preferences in a game or provide data for a visualization. Whatever the focus of your program is, you'll store the information users provide in data structures such as lists and dictionaries. When users close a program, you'll almost always want to save the information they entered. A simple way to do this involves storing your data using the `json` module.

El módulo `json` te permite convertir estructuras de datos sencillas de Python en cadenas con formato JSON, y luego cargar los datos desde ese archivo la próxima vez que se ejecute el programa. También puedes utilizar `json` para compartir datos entre distintos programas Python. Mejor aún, el formato de datos JSON no es específico de Python, por lo que puedes compartir los datos que almacenes en formato JSON con personas que trabajen en muchos otros lenguajes de programación. Es un formato útil y portátil, y es fácil de aprender.

The `json` module allows you to convert simple Python data structures into JSON-formatted strings, and then load the data from that file the next time the program runs. You can also use `json` to share data between different Python programs. Even better, the JSON data format is not specific to Python, so you can share data you store in the JSON format with people who work in many other programming languages. It's a useful and portable format, and it's easy to learn.

Nota (Note)

El formato *JSON (JavaScript Object Notation)* se desarrolló originalmente para JavaScript. Sin embargo, desde entonces se ha convertido en un formato común utilizado por muchos lenguajes, incluido Python.

The *JSON (JavaScript Object Notation)* format was originally developed for JavaScript. However, it has since become a common format used by many languages, including Python.

Utilización de `json.dumps()` y `json.loads()` (*Using `json.dumps()` and `json.loads()`*)

Vamos a escribir un breve programa que almacene un conjunto de números y otro programa que vuelva a leer estos números en la memoria. El primer programa utilizará `json.dumps()` para almacenar el conjunto de números, y el segundo programa utilizará `json.loads()` .

Let's write a short program that stores a set of numbers and another program that reads these numbers back into memory. The first program will use `json.dumps()` to store the set of numbers, and the second program will use `json.loads()` .

La función `json.dumps()` toma un argumento: un dato que debe convertirse al formato JSON. La función devuelve una cadena, que podemos escribir en un archivo de datos:

The `json.dumps()` function takes one argument: a piece of data that should be converted to the JSON format. The function returns a string, which we can then write to a data file:

numero_escritor.py

number_writer.py

```
from pathlib import Path
import json
```

```
numbers = [2, 3, 5, 7, 11, 13]
```

```
❶ path = Path('numbers.json')
❷ contents = json.dumps(numbers)
path.write_text(contents)
```

Primero importamos el módulo `json` , y luego creamos una lista de números con los que trabajar. A continuación, elegimos un nombre de archivo en el que almacenar la lista de números ❶ . Es habitual utilizar la extensión de archivo `.json` para indicar que los datos

del archivo se almacenan en formato JSON. A continuación, utilizamos la función `json.dumps()` ❷ para generar una cadena que contenga la representación JSON de los datos con los que estamos trabajando. Una vez que tenemos esta cadena, la escribimos en el archivo utilizando el mismo método `write_text()` que utilizamos antes.

We first import the `json` module, and then create a list of numbers to work with. Then we choose a filename in which to store the list of numbers ❶. It's customary to use the file extension `.json` to indicate that the data in the file is stored in the JSON format. Next, we use the `json.dumps()` ❷ function to generate a string containing the JSON representation of the data we're working with. Once we have this string, we write it to the file using the same `write_text()` method we used earlier.

Este programa no tiene salida, pero abramos el archivo *numbers.json* y echémosle un vistazo. Los datos se almacenan en un formato parecido al de Python:

This program has no output, but let's open the file *numbers.json* and look at it. The data is stored in a format that looks just like Python:

```
[2, 3, 5, 7, 11, 13]
```

Ahora escribiremos un programa independiente que utilice `json.loads()` para volver a leer la lista en memoria:

Now we'll write a separate program that uses `json.loads()` to read the list back into memory:

lector_de_numeros.py

number_reader.py

```
from pathlib import Path
import json
```

```
❶ path = Path('numbers.json')
❷ contents = path.read_text()
❸ numbers = json.loads(contents)
```

```
print(numbers)
```

Nos aseguramos de leer desde el mismo archivo que escribimos en ❶ . Como el archivo de datos es sólo un archivo de texto con un formato específico, podemos leerlo con el método `read_text()` ❷ . A continuación, pasamos el contenido del archivo a `json.loads()` ❸ . Esta función recibe una cadena con formato JSON y devuelve un objeto Python (en este caso, una lista), que asignamos a `numbers` . Por último, imprimimos la lista de números recuperada y vemos que es la misma lista creada en *number_writer.py* :

We make sure to read from the same file we wrote to ❶ . Since the data file is just a text file with specific formatting, we can read it with the `read_text()` method ❷ . We then pass the contents of the file to `json.loads()` ❸ . This function takes in a JSON-formatted string and returns a Python object (in this case, a list), which we assign to `numbers` . Finally, we print the recovered list of numbers and see that it's the same list created in *number_writer.py* :

```
[2, 3, 5, 7, 11, 13]
```

Ésta es una forma sencilla de compartir datos entre dos programas.

This is a simple way to share data between two programs.

Guardar y leer datos generados por el usuario (*Saving and Reading User-Generated Data*)

Guardar datos con `json` es útil cuando trabajas con datos generados por el usuario, porque si no almacenas la información de tu usuario de alguna manera, la perderás cuando el programa deje de ejecutarse. Veamos un ejemplo en el que pedimos al usuario su nombre la primera vez que ejecuta un programa y luego recordamos su nombre cuando vuelva a ejecutar el programa.

Saving data with `json` is useful when you're working with user-generated data, because if you don't store your user's information somehow, you'll lose it when the program stops running. Let's look at an example where we prompt the user for their name the first time they run a program and then remember their name when they run the program again.

Empecemos almacenando el nombre del usuario:

Let's start by storing the user's name:

recuérdame.py

remember_me.py

```
from pathlib import Path
import json
```

```
❶ username = input("What is your name? ")

❷ path = Path('username.json')
contents = json.dumps(username)
path.write_text(contents)

❸ print(f"We'll remember you when you come back, {username}!")
```

Primero pedimos un nombre de usuario para almacenarlo ❶ . A continuación, escribimos los datos que acabamos de recoger en un archivo llamado *username.json* ❷ . A continuación, imprimimos un mensaje informando al usuario de que hemos almacenado su información ❸ :

We first prompt for a username to store ❶ . Next, we write the data we just collected to a file called *username.json* ❷ . Then we print a message informing the user that we've stored their information ❸ :

```
What is your name? Eric
We'll remember you when you come back, Eric!
```

Ahora escribamos un nuevo programa que salude a un usuario cuyo nombre ya ha sido almacenado:

Now let's write a new program that greets a user whose name has already been stored:

saluda_usuario.py

greet_user.py

```
from pathlib import Path
import json

❶ path = Path('username.json')
contents = path.read_text()
❷ username = json.loads(contents)

print(f"Welcome back, {username}!")
```

Leemos el contenido del archivo de datos ❶ y luego utilizamos `json.loads()` para asignar los datos recuperados a la variable `username` ❷. Como hemos recuperado el nombre de usuario, podemos darle la bienvenida con un saludo personalizado:

We read the contents of the data file ❶ and then use `json.loads()` to assign the recovered data to the variable `username` ❷. Since we've recovered the username, we can welcome the user back with a personalized greeting:

Welcome back, Eric!

Necesitamos combinar estos dos programas en un único archivo. Cuando alguien ejecute *remember_me.py*, queremos recuperar su nombre de usuario de la memoria si es posible; si no, le pediremos un nombre de usuario y lo almacenaremos en *username.json* para la próxima vez. Podríamos escribir aquí un bloque `try - except` para responder adecuadamente si *username.json* no existe, pero en su lugar utilizaremos un práctico método del módulo `pathlib`:

We need to combine these two programs into one file. When someone runs *remember_me.py*, we want to retrieve their username from memory if possible; if not, we'll prompt for a username and store it in *username.json* for next time. We could write a `try -`

except block here to respond appropriately if *username.json* doesn't exist, but instead we'll use a handy method from the `pathlib` module:

recuérdame.py

remember_me.py

```
from pathlib import Path
import json
```

```
path = Path('username.json')
```

```
❶ if path.exists():
```

```
    contents = path.read_text()
```

```
    username = json.loads(contents)
```

```
    print(f"Welcome back, {username}!")
```

```
❷ else:
```

```
    username = input("What is your name? ")
```

```
    contents = json.dumps(username)
```

```
    path.write_text(contents)
```

```
    print(f"We'll remember you when you come back, {username}!")
```

Hay muchos métodos útiles que puedes utilizar con los objetos `Path`. El método `exists()` devuelve `True` si existe un archivo o carpeta y `False` si no existe. Aquí utilizamos `path.exists()` para averiguar si un nombre de usuario ya ha sido almacenado ❶. Si *username.json* existe, cargamos el nombre de usuario e imprimimos un saludo personalizado al usuario.

There are many helpful methods you can use with `Path` objects. The `exists()` method returns `True` if a file or folder exists and `False` if it doesn't. Here we use `path.exists()` to find out if a username has already been stored ❶. If *username.json* exists, we load the username and print a personalized greeting to the user.

Si el archivo *username.json* no existe ❷, pedimos un nombre de usuario y almacenamos el valor que introduzca el usuario. También imprimimos el mensaje familiar de que nos acordaremos de ellos cuando vuelvan.

If the file `username.json` doesn't exist ❷ , we prompt for a username and store the value that the user enters. We also print the familiar message that we'll remember them when they come back.

Sea cual sea el bloque que se ejecute, el resultado es un nombre de usuario y un saludo apropiado. Si es la primera vez que se ejecuta el programa, ésta es la salida:

Whichever block executes, the result is a username and an appropriate greeting. If this is the first time the program runs, this is the output:

What is your name? Eric
We'll remember you when you come back, Eric!

Si no:

Otherwise:

Welcome back, Eric!

Esta es la salida que ves si el programa ya se ha ejecutado al menos una vez. Aunque los datos de esta sección son sólo una cadena, el programa funcionaría igual de bien con cualquier dato que pueda convertirse a una cadena con formato JSON.

This is the output you see if the program was already run at least once. Even though the data in this section is just a single string, the program would work just as well with any data that can be converted to a JSON-formatted string.

Refactorización de (*Refactoring*)

A menudo, llegarás a un punto en el que tu código funcionará, pero reconocerás que podrías mejorarlo dividiéndolo en una serie de funciones que tengan tareas específicas. Este proceso se llama *refactoring* . La refactorización hace que tu código sea más limpio, más fácil de entender y más fácil de ampliar.

Often, you'll come to a point where your code will work, but you'll recognize that you could improve the code by breaking it up into a series of functions that have specific jobs. This process is called *refactoring*. Refactoring makes your code cleaner, easier to understand, and easier to extend.

Podemos refactorizar *remember_me.py* trasladando la mayor parte de su lógica a una o más funciones. El objetivo de *remember_me.py* es saludar al usuario, así que traslademos todo nuestro código existente a una función llamada `greet_user()` :

We can refactor *remember_me.py* by moving the bulk of its logic into one or more functions. The focus of *remember_me.py* is on greeting the user, so let's move all of our existing code into a function called `greet_user()` :

recuérdame.py

remember_me.py

```
from pathlib import Path
import json

def greet_user():
    ❶ """Greet the user by name."""
    path = Path('username.json')
    if path.exists():
        contents = path.read_text()
        username = json.loads(contents)
        print(f>Welcome back, {username}!")
    else:
        username = input("What is your name? ")
        contents = json.dumps(username)
        path.write_text(contents)
        print(f>We'll remember you when you come back, {username}!")

greet_user()
```

Como ahora estamos utilizando una función, reescribimos los comentarios como un docstring que refleja cómo funciona actualmente el programa ❶ . Este archivo es un poco

más limpio, pero la función `greet_user()` hace algo más que saludar al usuario: también recupera un nombre de usuario almacenado si existe y solicita uno nuevo si no existe.

Because we're using a function now, we rewrite the comments as a docstring that reflects how the program currently works ❶. This file is a little cleaner, but the function `greet_user()` is doing more than just greeting the user—it's also retrieving a stored username if one exists and prompting for a new username if one doesn't.

Vamos a refactorizar `greet_user()` para que no realice tantas tareas diferentes. Empezaremos moviendo el código para recuperar un nombre de usuario almacenado a una función separada:

Let's refactor `greet_user()` so it's not doing so many different tasks. We'll start by moving the code for retrieving a stored username to a separate function:

```
from pathlib import Path
import json

def get_stored_username(path):
    ❶ """Get stored username if available."""
    if path.exists():
        contents = path.read_text()
        username = json.loads(contents)
        return username
    else:
    ❷     return None

def greet_user():
    """Greet the user by name."""
    path = Path('username.json')
    username = get_stored_username(path)
    ❸ if username:
        print(f>Welcome back, {username}!")
    else:
        username = input("What is your name? ")
        contents = json.dumps(username)
        path.write_text(contents)
        print(f>We'll remember you when you come back, {username}!")
```

`greet_user()`

La nueva función `get_stored_username()` ❶ tiene un propósito claro, como se indica en el docstring. Esta función recupera un nombre de usuario almacenado y lo devuelve si lo encuentra. Si la ruta que se pasa a `get_stored_username()` no existe, la función devuelve `None` ❷. Esto es una buena práctica: una función debe devolver el valor que esperas o debe devolver `None`. Esto nos permite realizar una sencilla prueba con el valor de retorno de la función. Imprimimos un mensaje de bienvenida al usuario si el intento de recuperar un nombre de usuario tiene éxito ❸, y si no lo tiene, le pedimos un nuevo nombre de usuario.

The new function `get_stored_username()` ❶ has a clear purpose, as stated in the docstring. This function retrieves a stored username and returns the username if it finds one. If the path that's passed to `get_stored_username()` doesn't exist, the function returns `None` ❷. This is good practice: a function should either return the value you're expecting, or it should return `None`. This allows us to perform a simple test with the return value of the function. We print a welcome back message to the user if the attempt to retrieve a username is successful ❸, and if it isn't, we prompt for a new username.

Deberíamos eliminar un bloque más de código de `greet_user()`. Si el nombre de usuario no existe, deberíamos trasladar el código que solicita un nuevo nombre de usuario a una función dedicada a ese fin:

We should factor one more block of code out of `greet_user()`. If the username doesn't exist, we should move the code that prompts for a new username to a function dedicated to that purpose:

```
from pathlib import Path
import json
```

```
def get_stored_username(path):
    """Get stored username if available."""
    --snip--
```

```
def get_new_username(path):
    """Prompt for a new username."""
    username = input("What is your name? ")
    contents = json.dumps(username)
```

```
path.write_text(contents)
return username
```

```
def greet_user():
    """Greet the user by name."""
    path = Path('username.json')
    ❶ username = get_stored_username(path)
    if username:
        print(f>Welcome back, {username}!")
    else:
    ❷ username = get_new_username(path)
        print(f>We'll remember you when you come back, {username}!")

greet_user()
```

Cada función de esta versión final de *remember_me.py* tiene un único y claro propósito. Llamamos a `greet_user()` , y esa función imprime un mensaje apropiado: o bien da la bienvenida a un usuario existente o bien saluda a un nuevo usuario. Para ello, llama a `get_stored_username()` ❶ , que sólo se encarga de recuperar un nombre de usuario almacenado, si existe. Por último, si es necesario, `greet_user()` llama a `get_new_username()` ❷ , que sólo se encarga de obtener un nuevo nombre de usuario y almacenarlo. Esta compartimentación del trabajo es una parte esencial para escribir un código claro que sea fácil de mantener y ampliar.

Each function in this final version of *remember_me.py* has a single, clear purpose. We call `greet_user()` , and that function prints an appropriate message: it either welcomes back an existing user or greets a new user. It does this by calling `get_stored_username()` ❶ , which is responsible only for retrieving a stored username if one exists. Finally, if necessary, `greet_user()` calls `get_new_username()` ❷ , which is responsible only for getting a new username and storing it. This compartmentalization of work is an essential part of writing clear code that will be easy to maintain and extend.

Pruébalo tú mismo (*Try It Yourself*)

10-11. Número favorito: Escribe un programa que pregunte por el número favorito del usuario. Utiliza `json.dumps()` para almacenar este número en un archivo. Escribe otro programa que lea este valor e imprima el mensaje "¡Sé cuál es tu número favorito! Es ____".

10-11. Favorite Number: Write a program that prompts for the user's favorite number. Use `json.dumps()` to store this number in a file. Write a separate program that reads in this value and prints the message "I know your favorite number! It's ____."

10-12. Número Favorito Recordado: Combina los dos programas que escribiste en el Ejercicio 10-11 en un solo archivo. Si el número ya está almacenado, informa del número favorito al usuario. Si no, pregunta al usuario por su número favorito y guárdalo en un archivo. Ejecuta el programa dos veces para comprobar que funciona.

10-12. Favorite Number Remembered: Combine the two programs you wrote in Exercise 10-11 into one file. If the number is already stored, report the favorite number to the user. If not, prompt for the user's favorite number and store it in a file. Run the program twice to see that it works.

10-13. Diccionario del usuario: El ejemplo de *remember_me.py* sólo almacena un dato, el nombre de usuario. Amplía este ejemplo pidiendo dos datos más sobre el usuario, y almacena toda la información que recopiles en un diccionario. Escribe este diccionario en un archivo utilizando `json.dumps()` , y vuelve a leerlo utilizando `json.loads()` . Imprime un resumen que muestre exactamente lo que tu programa recuerda sobre el usuario.

10-13. User Dictionary: The *remember_me.py* example only stores one piece of information, the username. Expand this example by asking for two more pieces of information about the user, then store all the information you collect in a dictionary. Write this dictionary to a file using `json.dumps()` , and read it back in using `json.loads()` . Print a summary showing exactly what your program remembers about the user.

10-14. Verificar usuario: El listado final de *remember_me.py* asume que el usuario ya ha introducido su nombre de usuario o que el programa se está ejecutando por primera vez. Deberíamos modificarlo en caso de que el usuario actual no sea la persona que utilizó el programa por última vez.

10-14. Verify User: The final listing for *remember_me.py* assumes either that the user has already entered their username or that the program is running for the first time. We should modify it in case the current user is not the person who last used the program.

Antes de imprimir un mensaje de bienvenida en `greet_user()` , pregunta al usuario si éste es el nombre de usuario correcto. Si no lo es, llama a `get_new_username()` para obtener el nombre de usuario correcto.

Before printing a welcome back message in `greet_user()` , ask the user if this is the correct username. If it's not, call `get_new_username()` to get the correct username.

Resumen (Summary)

En este capítulo has aprendido a trabajar con archivos. Aprendiste a leer todo el contenido de un archivo, y luego a trabajar con el contenido línea a línea si es necesario. Aprendiste a escribir todo el texto que quieras en un archivo. También leíste sobre excepciones y sobre cómo manejar las excepciones que probablemente verás en tus programas. Por último, aprendiste a almacenar estructuras de datos Python para poder guardar la información que te proporcionen tus usuarios, evitando que tengan que empezar de nuevo cada vez que ejecuten un programa.

In this chapter, you learned how to work with files. You learned to read the entire contents of a file, and then work through the contents one line at a time if you need to. You learned to write as much text as you want to a file. You also read about exceptions and how to handle the exceptions you're likely to see in your programs. Finally, you learned how to store Python data structures so you can save information your users provide, preventing them from having to start over each time they run a program.

En el Capítulo 11, aprenderás formas eficaces de probar tu código. Esto te ayudará a confiar en que el código que desarrollas es correcto, y te ayudará a identificar los errores que se introduzcan a medida que sigas construyendo sobre los programas que has escrito.

In Chapter 11, you'll learn efficient ways to test your code. This will help you trust that the code you develop is correct, and it will help you identify bugs that are introduced as you continue to build on the programs you've written.

Capítulo 11: Probar tu código (Chapter 11: Testing Your Code)

11

Probar tu código (11 Testing Your Code)



DESCRIPCIÓN DE LA
IMAGEN (En español):

La imagen muestra una ilustración en blanco y negro de una serpiente con colmillos, envuelta alrededor de lo que parece ser el motor de un avión. La serpiente tiene una expresión decidida y los colmillos están visibles. El motor está detallado con aspas al frente. El fondo es de un color gris uniforme.

Cuando escribes una función o una clase, también puedes escribir pruebas para ese código. Las pruebas demuestran que tu código funciona como se supone que debe hacerlo en respuesta a todos los tipos de entrada que está diseñado para recibir. Cuando escribes pruebas, puedes estar seguro de que tu código funcionará correctamente cuando más gente empiece a utilizar tus programas. También podrás probar el código nuevo a medida que lo añadas, para asegurarte de que los cambios no rompen el comportamiento existente del programa. Todo programador comete errores, por lo que debe probar su código a menudo, para detectar los problemas antes de que los usuarios se encuentren con ellos.

When you write a function or a class, you can also write tests for that code. Testing proves that your code works as it's supposed to in response to all the kinds of input it's designed to receive. When you write tests, you can be confident that your code will work correctly as more people begin to use your programs. You'll also be able to test new code as you add it, to make sure your changes don't break your program's existing behavior. Every programmer makes mistakes, so every programmer must test their code often, to catch problems before users encounter them.

En este capítulo, aprenderás a probar tu código utilizando `pytest`. La biblioteca `pytest` es una colección de herramientas que te ayudarán a escribir tus primeras pruebas de forma

rápida y sencilla, a la vez que apoyan tus pruebas a medida que crecen en complejidad junto con tus proyectos. Python no incluye `pytest` por defecto, así que aprenderás a instalar bibliotecas externas. Saber instalar bibliotecas externas pondrá a tu disposición una gran variedad de código bien diseñado. Estas bibliotecas ampliarán enormemente el tipo de proyectos en los que puedes trabajar.

In this chapter, you'll learn to test your code using `pytest`. The `pytest` library is a collection of tools that will help you write your first tests quickly and simply, while supporting your tests as they grow in complexity along with your projects. Python doesn't include `pytest` by default, so you'll learn to install external libraries. Knowing how to install external libraries will make a wide variety of well-designed code available to you. These libraries will expand the kinds of projects you can work on immensely.

Aprenderás a construir una serie de pruebas y a comprobar que cada conjunto de entradas da como resultado la salida que desees. Verás qué aspecto tiene una prueba superada y qué aspecto tiene una prueba fallida, y aprenderás cómo una prueba fallida puede ayudarte a mejorar tu código. Aprenderás a probar funciones y clases, y empezarás a comprender cuántas pruebas debes escribir para un proyecto.

You'll learn to build a series of tests and check that each set of inputs results in the output you want. You'll see what a passing test looks like and what a failing test looks like, and you'll learn how a failing test can help you improve your code. You'll learn to test functions and classes, and you'll start to understand how many tests to write for a project.

Instalar `pytest` con `pip` (*Installing `pytest` with `pip`*)

Aunque Python incluye mucha funcionalidad en la biblioteca estándar, los desarrolladores de Python también dependen en gran medida de paquetes de terceros. Un *third-party package* es una biblioteca que se desarrolla fuera del núcleo del lenguaje Python. Algunas bibliotecas populares de terceros acaban siendo adoptadas por la biblioteca estándar, y terminan incluyéndose en la mayoría de las instalaciones de Python a partir de ese momento. Esto ocurre sobre todo con las bibliotecas que probablemente no cambien mucho una vez resueltos sus errores iniciales. Este tipo de bibliotecas pueden evolucionar al mismo ritmo que el lenguaje en general.

While Python includes a lot of functionality in the standard library, Python developers also depend heavily on third-party packages. A *third-party package* is a library that's developed outside the core Python language. Some popular third-party libraries are eventually

adopted into the standard library, and end up being included in most Python installations from that point forward. This happens most often with libraries that are unlikely to change much once they've had their initial bugs worked out. These kinds of libraries can evolve at the same pace as the overall language.

Muchos paquetes, sin embargo, se mantienen fuera de la biblioteca estándar para que puedan desarrollarse a un ritmo independiente del propio lenguaje. Estos paquetes tienden a actualizarse con más frecuencia de lo que lo harían si estuvieran ligados al calendario de desarrollo de Python. Este es el caso de `pytest` y de la mayoría de las bibliotecas que utilizaremos en la segunda mitad de este libro. No debes confiar ciegamente en todos los paquetes de terceros, pero tampoco debes desanimarte por el hecho de que muchas funcionalidades importantes se implementan a través de dichos paquetes.

Many packages, however, are kept out of the standard library so they can be developed on a timeline independent of the language itself. These packages tend to be updated more frequently than they would be if they were tied to Python's development schedule. This is true of `pytest` and most of the libraries we'll use in the second half of this book. You shouldn't blindly trust every third-party package, but you also shouldn't be put off by the fact that a lot of important functionality is implemented through such packages.

Actualizar pip (*Updating pip*)

Python incluye una herramienta llamada `pip` que se utiliza para instalar paquetes de terceros. Como `pip` ayuda a instalar paquetes de recursos externos, se actualiza a menudo para solucionar posibles problemas de seguridad. Así que empezaremos por actualizar `pip`.

Python includes a tool called `pip` that's used to install third-party packages. Because `pip` helps install packages from external resources, it's updated often to address potential security issues. So, we'll start by updating `pip`.

Abre una nueva ventana de terminal y ejecuta el siguiente comando:

Open a new terminal window and issue the following command:

```
$ python -m pip install --upgrade pip
```

❶ Requirement already satisfied: pip in /.../python3.11/site-packages (22.0.4)
--snip--

❷ Successfully installed pip-22.1.2

La primera parte de este comando, `python -m pip`, indica a Python que ejecute el módulo `pip`. La segunda parte, `install --upgrade`, indica a `pip` que actualice un paquete ya instalado. La última parte, `pip`, especifica qué paquete de terceros debe actualizarse. La salida muestra que mi versión actual de `pip`, la 22.0.4 ❶, fue sustituida por la última versión en el momento de escribir esto, la 22.1.2 ❷.

The first part of this command, `python -m pip`, tells Python to run the module `pip`. The second part, `install --upgrade`, tells `pip` to update a package that's already been installed. The last part, `pip`, specifies which third-party package should be updated. The output shows that my current version of `pip`, version 22.0.4 ❶, was replaced by the latest version at the time of this writing, 22.1.2 ❷.

Puedes utilizar este comando para actualizar cualquier paquete de terceros instalado en tu sistema:

You can use this command to update any third-party package installed on your system:

```
$ python -m pip install --upgrade package_name
```

Nota (Note)

Si utilizas Linux, es posible que `pip` no esté incluido en tu instalación de Python. Si obtienes un error al intentar actualizar `pip`, consulta las instrucciones del Apéndice A.

If you're using Linux, `pip` may not be included with your installation of Python. If you get an error when trying to upgrade `pip`, see the instructions in Appendix A.

Instalar pytest (Installing pytest)

Ahora que pip está actualizado, podemos instalar `pytest` :

Now that pip is up to date, we can install `pytest` :

```
$ python -m pip install --user pytest
Collecting pytest
--snip--
Successfully installed attrs-21.4.0 iniconfig-1.1.1 ...pytest-7.x.x
```

Seguimos utilizando el comando principal `pip install` , sin la bandera `--upgrade` esta vez. En su lugar, utilizaremos la bandera `--user` , que indica a Python que instale este paquete sólo para el usuario actual. La salida muestra que se ha instalado correctamente la última versión de `pytest` , junto con otros paquetes de los que depende `pytest` .

We're still using the core command `pip install` , without the `--upgrade` flag this time. Instead, we're using the `--user` flag, which tells Python to install this package for the current user only. The output shows that the latest version of `pytest` was successfully installed, along with a number of other packages that `pytest` depends on.

Puedes utilizar este comando para instalar muchos paquetes de terceros:

You can use this command to install many third-party packages:

```
$ python -m pip install --user package_name
```

Nota (Note)

Si tienes alguna dificultad para ejecutar este comando, intenta ejecutar el mismo comando sin la bandera `--user` .

If you have any difficulty running this command, try running the same command without the `--user` flag.

Probar una función (*Testing a Function*)

Para aprender a probar, necesitamos código que probar. Aquí tienes una función sencilla que recibe un nombre y un apellido y devuelve un nombre completo con un formato claro:

To learn about testing, we need code to test. Here's a simple function that takes in a first and last name, and returns a neatly formatted full name:

nombre_funcion.py

name_function.py

```
def get_formatted_name(first, last):
    """Generate a neatly formatted full name."""
    full_name = f'{first} {last}'
    return full_name.title()
```

La función `get_formatted_name()` combina el nombre y los apellidos con un espacio intermedio para completar un nombre completo, y luego escribe en mayúsculas y devuelve el nombre completo. Para comprobar que `get_formatted_name()` funciona, vamos a crear un programa que utilice esta función. El programa *names.py* permite a los usuarios introducir un nombre y un apellido, y ver un nombre completo formateado de forma clara:

The function `get_formatted_name()` combines the first and last name with a space in between to complete a full name, and then capitalizes and returns the full name. To check that `get_formatted_name()` works, let's make a program that uses this function. The program *names.py* lets users enter a first and last name, and see a neatly formatted full name:

nombres.py

names.py

```
from name_function import get_formatted_name

print("Enter 'q' at any time to quit.")
while True:
    first = input("\nPlease give me a first name: ")
    if first == 'q':
        break
    last = input("Please give me a last name: ")
    if last == 'q':
        break

    formatted_name = get_formatted_name(first, last)
    print(f"\tNeatly formatted name: {formatted_name}.")
```

Este programa importa `get_formatted_name()` de *name_function.py*. El usuario puede introducir una serie de nombres y apellidos y ver los nombres completos formateados que se generan:

This program imports `get_formatted_name()` from *name_function.py*. The user can enter a series of first and last names and see the formatted full names that are generated:

Enter 'q' at any time to quit.

```
Please give me a first name: janis
Please give me a last name: joplin
    Neatly formatted name: Janis Joplin.
```

```
Please give me a first name: bob
Please give me a last name: dylan
    Neatly formatted name: Bob Dylan.
```

```
Please give me a first name: q
```

Podemos ver que los nombres generados aquí son correctos. Pero digamos que queremos modificar `get_formatted_name()` para que también pueda manejar segundos nombres. Al

hacerlo, queremos asegurarnos de que no rompemos la forma en que la función gestiona los nombres que sólo tienen nombre y apellidos. Podríamos probar nuestro código ejecutando *names.py* e introduciendo un nombre como Janis Joplin cada vez que modificamos `get_formatted_name()`, pero eso se volvería tedioso. Afortunadamente, `pytest` proporciona una forma eficaz de automatizar la comprobación de la salida de una función. Si automatizamos las pruebas de `get_formatted_name()`, siempre podremos estar seguros de que la función funcionará cuando se le den los tipos de nombres para los que hemos escrito pruebas.

We can see that the names generated here are correct. But say we want to modify `get_formatted_name()` so it can also handle middle names. As we do so, we want to make sure we don't break the way the function handles names that have only a first and last name. We could test our code by running *names.py* and entering a name like Janis Joplin every time we modify `get_formatted_name()`, but that would become tedious. Fortunately, `pytest` provides an efficient way to automate the testing of a function's output. If we automate the testing of `get_formatted_name()`, we can always be confident that the function will work when given the kinds of names we've written tests for.

Pruebas unitarias y casos de prueba (*Unit Tests and Test Cases*)

Existe una gran variedad de enfoques para probar el software. Uno de los tipos de prueba más sencillos es la prueba unitaria. Una *unit test* verifica que un aspecto concreto del comportamiento de una función es correcto. Un *test case* es una colección de pruebas unitarias que, en conjunto, demuestran que una función se comporta como se supone que debe hacerlo, dentro de toda la gama de situaciones que esperas que maneje.

There is a wide variety of approaches to testing software. One of the simplest kinds of test is a unit test. A *unit test* verifies that one specific aspect of a function's behavior is correct. A *test case* is a collection of unit tests that together prove that a function behaves as it's supposed to, within the full range of situations you expect it to handle.

Un buen caso de prueba tiene en cuenta todos los tipos posibles de entrada que podría recibir una función e incluye pruebas para representar cada una de estas situaciones. Un caso de prueba con *full coverage* incluye una gama completa de pruebas unitarias que cubren todas las formas posibles de utilizar una función. Conseguir una cobertura completa en un proyecto grande puede ser desalentador. A menudo es suficiente con escribir pruebas para los comportamientos críticos de tu código y luego aspirar a una cobertura total sólo si el proyecto empieza a tener un uso generalizado.

A good test case considers all the possible kinds of input a function could receive and includes tests to represent each of these situations. A test case with *full coverage* includes a full range of unit tests covering all the possible ways you can use a function. Achieving full coverage on a large project can be daunting. It's often good enough to write tests for your code's critical behaviors and then aim for full coverage only if the project starts to see widespread use.

Una prueba que pasa (A Passing Test)

Con `pytest`, escribir tu primera prueba unitaria es bastante sencillo. Escribiremos una única función de prueba. La función de prueba llamará a la función que estamos probando y haremos una afirmación sobre el valor devuelto. Si nuestra afirmación es correcta, la prueba pasará; si la afirmación es incorrecta, la prueba fallará.

With `pytest`, writing your first unit test is pretty straightforward. We'll write a single test function. The test function will call the function we're testing, and we'll make an assertion about the value that's returned. If our assertion is correct, the test will pass; if the assertion is incorrect, the test will fail.

Aquí tienes la primera prueba de la función `get_formatted_name()` :

Here's the first test of the function `get_formatted_name()` :

nombre_prueba_funcion.py

test_name_function.py

```
from name_function import get_formatted_name
```

```
❶ def test_first_last_name():  
    """Do names like 'Janis Joplin' work?"""
```

- ❷ `formatted_name = get_formatted_name('janis', 'joplin')`
- ❸ `assert formatted_name == 'Janis Joplin'`

Antes de ejecutar la prueba, veamos más detenidamente esta función. El nombre del archivo de prueba es importante; debe empezar por `test_`. Cuando le pidamos a `pytest` que ejecute las pruebas que hemos escrito, buscará cualquier archivo que empiece por `test_`, y ejecutará todas las pruebas que encuentre en ese archivo.

Before we run the test, let's take a closer look at this function. The name of a test file is important; it must start with `test_`. When we ask `pytest` to run the tests we've written, it will look for any file that begins with `test_`, and run all of the tests it finds in that file.

En el archivo de prueba, primero importamos la función que queremos probar: `get_formatted_name()`. Después definimos una función de prueba: en este caso, `test_first_last_name()` ❶. Se trata de un nombre de función más largo que el que hemos estado utilizando, por una buena razón. En primer lugar, las funciones de prueba deben empezar por la palabra `test`, seguida de un guión bajo. Cualquier función que empiece por `test_` será *discovered* por `pytest`, y se ejecutará como parte del proceso de prueba.

In the test file, we first import the function that we want to test: `get_formatted_name()`. Then we define a test function: in this case, `test_first_last_name()` ❶. This is a longer function name than we've been using, for a good reason. First, test functions need to start with the word `test`, followed by an underscore. Any function that starts with `test_` will be *discovered* by `pytest`, and will be run as part of the testing process.

Además, los nombres de las pruebas deben ser más largos y descriptivos que el nombre típico de una función. Nunca llamarás a la función tú mismo; `pytest` encontrará la función y la ejecutará por ti. Los nombres de las funciones de prueba deben ser lo suficientemente largos como para que, si ves el nombre de la función en un informe de prueba, tengas una buena idea de qué comportamiento se estaba probando.

Also, test names should be longer and more descriptive than a typical function name. You'll never call the function yourself; `pytest` will find the function and run it for you. Test function names should be long enough that if you see the function name in a test report, you'll have a good sense of what behavior was being tested.

A continuación, llamamos a la función que estamos probando ❷ . Aquí llamamos a `get_formatted_name()` con los argumentos `'janis'` y `'joplin'` , igual que cuando ejecutamos `names.py` . Asignamos el valor de retorno de esta función a `formatted_name` .

Next, we call the function we're testing ❷ . Here we call `get_formatted_name()` with the arguments `'janis'` and `'joplin'` , just like we used when we ran `names.py` . We assign the return value of this function to `formatted_name` .

Por último, hacemos una afirmación ❸ . Una *assertion* es una afirmación sobre una condición. Aquí afirmamos que el valor de `formatted_name` debe ser `'Janis Joplin'` .

Finally, we make an assertion ❸ . An *assertion* is a claim about a condition. Here we're claiming that the value of `formatted_name` should be `'Janis Joplin'` .

Ejecutar una prueba (*Running a Test*)

Si ejecutas directamente el archivo `test_name_function.py` , no obtendrás ninguna salida porque nunca hemos llamado a la función de prueba. En su lugar, haremos que `pytest` ejecute el archivo de prueba por nosotros.

If you run the file `test_name_function.py` directly, you won't get any output because we never called the test function. Instead, we'll have `pytest` run the test file for us.

Para ello, abre una ventana de terminal y navega hasta la carpeta que contiene el archivo de prueba. Si utilizas VS Code, puedes abrir la carpeta que contiene el archivo de prueba y utilizar el terminal que está incrustado en la ventana del editor. En la ventana del terminal, introduce el comando `pytest` . Esto es lo que deberías ver:

To do this, open a terminal window and navigate to the folder that contains the test file. If you're using VS Code, you can open the folder containing the test file and use the terminal that's embedded in the editor window. In the terminal window, enter the command `pytest` . Here's what you should see:

```
$ pytest
===== test session starts =====
❶ platform darwin -- Python 3.x.x, pytest-7.x.x, pluggy-1.x.x
❷ rootdir: ../../python_work/chapter_11
❸ collected 1 item

❹ test_name_function.py . [100%]
===== 1 passed in 0.00s =====
```

Intentemos dar sentido a esta salida. En primer lugar, vemos alguna información sobre el sistema en el que se está ejecutando la prueba ❶ . Lo estoy probando en un sistema macOS, por lo que es posible que veas algunos resultados diferentes. Lo más importante es que podemos ver qué versiones de Python, pytest , y otros paquetes se están utilizando para ejecutar la prueba.

Let's try to make sense of this output. First of all, we see some information about the system the test is running on ❶ . I'm testing this on a macOS system, so you may see some different output here. Most importantly, we can see which versions of Python, pytest , and other packages are being used to run the test.

A continuación, vemos el directorio en el que se está ejecutando la prueba desde ❷ : en mi caso, *python_work/chapter_11* . Podemos ver que pytest encontró una prueba para ejecutar ❸ , y podemos ver el archivo de prueba que se está ejecutando ❹ . El único punto tras el nombre del archivo nos indica que se ha superado una única prueba, y 100% deja claro que se han ejecutado todas las pruebas. Un proyecto grande puede tener cientos o miles de pruebas, y los puntos y el indicador de porcentaje completado pueden ser útiles para controlar el progreso general de la ejecución de las pruebas.

Next, we see the directory where the test is being run from ❷ : in my case, *python_work/chapter_11* . We can see that pytest found one test to run ❸ , and we can see the test file that's being run ❹ . The single dot after the name of the file tells us that a single test passed, and the 100% makes it clear that all of the tests have been run. A large project can have hundreds or thousands of tests, and the dots and percentage-complete indicator can be helpful in monitoring the overall progress of the test run.

La última línea nos indica que se ha superado una prueba y que se ha tardado menos de 0,01 segundos en ejecutarla.

The last line tells us that one test passed, and it took less than 0.01 seconds to run the test.

Esta salida indica que la función `get_formatted_name()` siempre funcionará para nombres que tengan nombre y apellidos, a menos que modifiquemos la función. Cuando modifiquemos `get_formatted_name()` , podremos volver a ejecutar esta prueba. Si la prueba se supera, sabremos que la función seguirá funcionando para nombres como Janis Joplin.

This output indicates that the function `get_formatted_name()` will always work for names that have a first and last name, unless we modify the function. When we modify `get_formatted_name()` , we can run this test again. If the test passes, we know the function will still work for names like Janis Joplin.

Nota (Note)

Si no estás seguro de cómo navegar hasta la ubicación correcta en el terminal, consulta " Ejecutar programas Python desde un terminal " en la página 11 . Además, si ves un mensaje de que no se ha encontrado el comando `pytest` , utiliza en su lugar el comando `python -m pytest` .

If you're not sure how to navigate to the right location in the terminal, see " Running Python Programs from a Terminal " on page 11 . Also, if you see a message that the `pytest` command was not found, use the command `python -m pytest` instead.

Una prueba fallida (A Failing Test)

¿Qué aspecto tiene una prueba que falla? Modifiquemos `get_formatted_name()` para que pueda manejar segundos nombres, pero hagámoslo de forma que rompa la función para nombres con sólo nombre y apellidos, como Janis Joplin.

What does a failing test look like? Let's modify `get_formatted_name()` so it can handle middle names, but let's do so in a way that breaks the function for names with just a first and last name, like Janis Joplin.

Aquí tienes una nueva versión de `get_formatted_name()` que requiere un argumento de segundo nombre:

Here's a new version of `get_formatted_name()` that requires a middle name argument:

nombre_funcion.py

name_function.py

```
def get_formatted_name(first, middle, last):
    """Generate a neatly formatted full name."""
    full_name = f'{first} {middle} {last}'
    return full_name.title()
```

Esta versión debería funcionar para las personas con segundo nombre, pero cuando la probamos, vemos que hemos roto la función para las personas que sólo tienen nombre y apellidos.

This version should work for people with middle names, but when we test it, we see that we've broken the function for people with just a first and last name.

Esta vez, al ejecutar `pytest` obtenemos el siguiente resultado:

This time, running `pytest` gives the following output:

```
$ pytest
===== test session starts =====
--snip--
❶ test_name_function.py F [100%]
❷ ===== FAILURES =====
❸ _____ test_first_last_name _____
```

```

def test_first_last_name():
    """Do names like 'Janis Joplin' work?"""
❷ > formatted_name = get_formatted_name('janis', 'joplin')
❸ E   TypeError: get_formatted_name() missing 1 required positional
    argument: 'last'

test_name_function.py:5: TypeError
===== short test summary info =====
FAILED test_name_function.py::test_first_last_name - TypeError:
  get_formatted_name() missing 1 required positional argument: 'last'
===== 1 failed in 0.04s =====

```

Hay mucha información aquí porque hay muchas cosas que puedes necesitar saber cuando falla una prueba. El primer elemento destacable de la salida es un único **F** ❶, que nos indica que ha fallado una prueba. A continuación vemos una sección que se centra en **FAILURES** ❷, porque las pruebas fallidas suelen ser lo más importante en lo que centrarse en una ejecución de prueba. A continuación, vemos que `test_first_last_name()` era la función de prueba que falló ❸. Un corchete angular ❹ indica la línea de código que provocó el fallo de la prueba. El **E** en la siguiente línea ❺ muestra el error real que causó el fallo: un `TypeError` debido a la falta de un argumento posicional requerido, `last`. La información más importante se repite en un resumen más breve al final, de modo que cuando ejecutes muchas pruebas, puedas hacerte una idea rápida de qué pruebas fallaron y por qué.

There's a lot of information here because there's a lot you might need to know when a test fails. The first item of note in the output is a single **F** ❶, which tells us that one test failed. We then see a section that focuses on **FAILURES** ❷, because failed tests are usually the most important thing to focus on in a test run. Next, we see that `test_first_last_name()` was the test function that failed ❸. An angle bracket ❹ indicates the line of code that caused the test to fail. The **E** on the next line ❺ shows the actual error that caused the failure: a `TypeError` due to a missing required positional argument, `last`. The most important information is repeated in a shorter summary at the end, so when you're running many tests, you can get a quick sense of which tests failed and why.

Responder a una prueba fallida (*Responding to a Failed Test*)

¿Qué haces cuando falla una prueba? Suponiendo que estés comprobando las condiciones correctas, una prueba superada significa que la función se comporta correctamente y una prueba fallida significa que hay un error en el nuevo código que has escrito. Así que cuando falle una prueba, no cambies la prueba. Si lo haces, puede que tus pruebas pasen, pero cualquier código que llame a tu función como lo hace la prueba dejará de funcionar de repente. En lugar de eso, arregla el código que está haciendo que falle la prueba. Examina

los cambios que acabas de hacer en la función, y averigua cómo esos cambios rompieron el comportamiento deseado.

What do you do when a test fails? Assuming you're checking the right conditions, a passing test means the function is behaving correctly and a failing test means there's an error in the new code you wrote. So when a test fails, don't change the test. If you do, your tests might pass, but any code that calls your function like the test does will suddenly stop working. Instead, fix the code that's causing the test to fail. Examine the changes you just made to the function, and figure out how those changes broke the desired behavior.

En este caso, `get_formatted_name()` solía requerir sólo dos parámetros: un nombre y un apellido. Ahora requiere un nombre, un segundo nombre y un apellido. La adición de ese parámetro obligatorio del segundo nombre rompió el comportamiento original de `get_formatted_name()`. La mejor opción en este caso es hacer que el segundo nombre sea opcional. Una vez que lo hagamos, nuestra prueba para nombres como Janis Joplin debería volver a pasar, y también deberíamos poder aceptar segundos nombres. Modifiquemos `get_formatted_name()` para que los segundos nombres sean opcionales y volvamos a ejecutar el caso de prueba. Si se supera, pasaremos a asegurarnos de que la función gestiona correctamente los segundos nombres.

In this case, `get_formatted_name()` used to require only two parameters: a first name and a last name. Now it requires a first name, middle name, and last name. The addition of that mandatory middle name parameter broke the original behavior of `get_formatted_name()`. The best option here is to make the middle name optional. Once we do, our test for names like Janis Joplin should pass again, and we should be able to accept middle names as well. Let's modify `get_formatted_name()` so middle names are optional and then run the test case again. If it passes, we'll move on to making sure the function handles middle names properly.

Para que los segundos nombres sean opcionales, movemos el parámetro `middle` al final de la lista de parámetros en la definición de la función y le damos un valor por defecto vacío. También añadimos una prueba `if` que construye el nombre completo correctamente, dependiendo de si se proporciona un segundo nombre:

To make middle names optional, we move the parameter `middle` to the end of the parameter list in the function definition and give it an empty default value. We also add an

if test that builds the full name properly, depending on whether a middle name is provided:

nombre_funcion.py

name_function.py

```
def get_formatted_name(first, last, middle=""):
    """Generate a neatly formatted full name."""
    if middle:
        full_name = f"{first} {middle} {last}"
    else:
        full_name = f"{first} {last}"
    return full_name.title()
```

En esta nueva versión de `get_formatted_name()` , el segundo nombre es opcional. Si se pasa un segundo nombre a la función, el nombre completo contendrá el nombre, el segundo nombre y los apellidos. En caso contrario, el nombre completo constará sólo de nombre y apellidos. Ahora la función debería funcionar para ambos tipos de nombres. Para averiguar si la función sigue funcionando para nombres como `Janis Joplin` , vamos a ejecutar de nuevo la prueba:

In this new version of `get_formatted_name()` , the middle name is optional. If a middle name is passed to the function, the full name will contain a first, middle, and last name. Otherwise, the full name will consist of just a first and last name. Now the function should work for both kinds of names. To find out if the function still works for names like `Janis Joplin` , let's run the test again:

```
$ pytest
===== test session starts =====
--snip--
test_name_function.py . [100%]
===== 1 passed in 0.00s =====
```

Ahora la prueba pasa. Esto es ideal; significa que la función vuelve a funcionar para nombres como `Janis Joplin` , sin que tengamos que probar la función manualmente. Arreglar nuestra función fue más fácil porque la prueba fallida nos ayudó a identificar cómo el nuevo código rompía el comportamiento existente.

The test passes now. This is ideal; it means the function works for names like Janis Joplin again, without us having to test the function manually. Fixing our function was easier because the failed test helped us identify how the new code broke existing behavior.

Añadir nuevas pruebas (Adding New Tests)

Ahora que sabemos que `get_formatted_name()` vuelve a funcionar con nombres sencillos, escribamos una segunda prueba para las personas que incluyen un segundo nombre. Para ello, añadimos otra función de prueba al archivo `test_name_function.py`:

Now that we know `get_formatted_name()` works for simple names again, let's write a second test for people who include a middle name. We do this by adding another test function to the file `test_name_function.py`:

función_nombre_prueba.py

test_name_function.py

```
from name_function import get_formatted_name
```

```
def test_first_last_name():  
    --snip--
```

```
def test_first_last_middle_name():  
    """Do names like 'Wolfgang Amadeus Mozart' work?"""  
    ❶ formatted_name = get_formatted_name(  
        'wolfgang', 'mozart', 'amadeus')  
    ❷ assert formatted_name == 'Wolfgang Amadeus Mozart'
```

Llamamos a esta nueva función `test_first_last_middle_name()`. El nombre de la función debe empezar por `test_` para que la función se ejecute automáticamente cuando ejecutemos `pytest`. Nombramos la función para que quede claro qué comportamiento de `get_formatted_name()` estamos probando. De este modo, si la prueba falla, sabremos inmediatamente qué tipo de nombres se ven afectados.

We name this new function `test_first_last_middle_name()` . The function name must start with `test_` so the function runs automatically when we run `pytest` . We name the function to make it clear which behavior of `get_formatted_name()` we're testing. As a result, if the test fails, we'll know right away what kinds of names are affected.

Para probar la función, llamamos a `get_formatted_name()` con un nombre, apellido y segundo nombre ❶ , y luego hacemos una afirmación ❷ de que el nombre completo devuelto coincide con el nombre completo (nombre, apellido y segundo nombre) que esperamos. Cuando volvemos a ejecutar `pytest` , ambas pruebas pasan:

To test the function, we call `get_formatted_name()` with a first, last, and middle name ❶ , and then we make an assertion ❷ that the returned full name matches the full name (first, middle, and last) that we expect. When we run `pytest` again, both tests pass:

```
$ pytest
===== test session starts =====
--snip--
collected 2 items

❶ test_name_function.py .. [100%]
===== 2 passed in 0.01s =====
```

Los dos puntos ❶ indican que se han superado dos pruebas, lo que también queda claro en la última línea de salida. ¡Esto es estupendo! Ahora sabemos que la función sigue funcionando para nombres como `Janis Joplin` , y podemos estar seguros de que también funcionará para nombres como `Wolfgang Amadeus Mozart` .

The two dots ❶ indicate that two tests passed, which is also clear from the last line of output. This is great! We now know that the function still works for names like `Janis Joplin` , and we can be confident that it will work for names like `Wolfgang Amadeus Mozart` as well.

Pruébalo tú mismo (Try It Yourself)

11-1. Ciudad, País: Escribe una función que acepte dos parámetros: el nombre de una ciudad y el nombre de un país. La función debe devolver una única cadena de la forma `City, Country` , como `Santiago, Chile` . Guarda la función en un módulo llamado `city_functions.py` , y guarda este archivo en una carpeta nueva para que `pytest` no intente ejecutar las pruebas que ya hemos escrito.

11-1. City, Country: Write a function that accepts two parameters: a city name and a country name. The function should return a single string of the form `City, Country` , such as `Santiago, Chile` . Store the function in a module called `city_functions.py` , and save this file in a new folder so `pytest` won't try to run the tests we've already written.

Crea un archivo llamado `test_cities.py` que compruebe la función que acabas de escribir. Escribe una función llamada `test_city_country()` para verificar que al llamar a tu función con valores como `'santiago'` y `'chile'` se obtiene la cadena correcta. Ejecuta la prueba y asegúrate de que `test_city_country()` pasa.

Create a file called `test_cities.py` that tests the function you just wrote. Write a function called `test_city_country()` to verify that calling your function with values such as `'santiago'` and `'chile'` results in the correct string. Run the test, and make sure `test_city_country()` passes.

11-2. Población: Modifica tu función para que requiera un tercer parámetro, `population` . Ahora debería devolver una única cadena de la forma `City, Country – population xxx` , como `Santiago, Chile – population 5000000` . Ejecuta de nuevo la prueba y asegúrate de que `test_city_country()` falla esta vez.

11-2. Population: Modify your function so it requires a third parameter, `population` . It should now return a single string of the form `City, Country – population xxx` , such as `Santiago, Chile – population 5000000` . Run the test again, and make sure `test_city_country()` fails this time.

Modifica la función para que el parámetro `population` sea opcional. Ejecuta la prueba y asegúrate de que `test_city_country()` vuelve a pasar.

Modify the function so the `population` parameter is optional. Run the test, and make sure `test_city_country()` passes again.

Escribe una segunda prueba llamada `test_city_country_population()` que verifique que puedes llamar a tu función con los valores `'santiago'`, `'chile'` y `'population=5000000'`. Ejecuta las pruebas una vez más, y asegúrate de que esta nueva prueba pasa.

Write a second test called `test_city_country_population()` that verifies you can call your function with the values `'santiago'`, `'chile'`, and `'population=5000000'`. Run the tests one more time, and make sure this new test passes.

Probar una clase (*Testing a Class*)

En la primera parte de este capítulo, escribiste pruebas para una única función. Ahora escribirás pruebas para una clase. Utilizarás clases en muchos de tus programas, por lo que es útil poder demostrar que tus clases funcionan correctamente. Si tienes pruebas que pasan para una clase en la que estás trabajando, puedes estar seguro de que las mejoras que hagas en la clase no romperán accidentalmente su comportamiento actual.

In the first part of this chapter, you wrote tests for a single function. Now you'll write tests for a class. You'll use classes in many of your own programs, so it's helpful to be able to prove that your classes work correctly. If you have passing tests for a class you're working on, you can be confident that improvements you make to the class won't accidentally break its current behavior.

Variedad de afirmaciones (*A Variety of Assertions*)

Hasta ahora sólo has visto un tipo de aserción: la afirmación de que una cadena tiene un valor determinado. Al escribir una prueba, puedes hacer cualquier afirmación que pueda expresarse como una declaración condicional. Si la condición es `True` como esperabas, se confirmará tu suposición sobre cómo se comporta esa parte de tu programa; puedes estar seguro de que no existen errores. Si la condición que supone que es `True` es en realidad `False`, la prueba fallará y sabrás que hay un problema que resolver. [La Tabla 11-1](#) muestra algunos de los tipos de aserciones más útiles que puedes incluir en tus pruebas iniciales.

So far, you’ve seen just one kind of assertion: a claim that a string has a specific value. When writing a test, you can make any claim that can be expressed as a conditional statement. If the condition is `True` as expected, your assumption about how that part of your program behaves will be confirmed; you can be confident that no errors exist. If the condition you assume is `True` is actually `False`, the test will fail and you’ll know there’s an issue to resolve. [Table 11-1](#) shows some of the most useful kinds of assertions you can include in your initial tests.

Tabla en español:	Afirmación	Table in English:	Assertion
	<code>assert a == b</code>		<code>assert a == b</code>
	<code>assert a != b</code>		<code>assert a != b</code>
	<code>assert a</code>		<code>assert a</code>
	<code>assert not a</code>		<code>assert not a</code>
	<code>assert element</code>		<code>assert element</code>
	<code>assert element</code>		<code>assert element</code>

Tabla 11-1 : Afirmaciones de uso común en las pruebas

Table 11-1 : Commonly Used Assertion Statements in Tests

Éstos son sólo algunos ejemplos; cualquier cosa que pueda expresarse como una declaración condicional puede incluirse en una prueba.

These are just a few examples; anything that can be expressed as a conditional statement can be included in a test.

Una clase para probar (A Class to Test)

Probar una clase es similar a probar una función, porque gran parte del trabajo consiste en probar el comportamiento de los métodos de la clase. Sin embargo, hay algunas diferencias, así que vamos a escribir una clase para probar. Considera una clase que ayude a administrar encuestas anónimas:

Testing a class is similar to testing a function, because much of the work involves testing the behavior of the methods in the class. However, there are a few differences, so let's write a class to test. Consider a class that helps administer anonymous surveys:

encuesta.py

survey.py

```
class AnonymousSurvey:
```

```
    """Collect anonymous answers to a survey question."""
```

- ❶

```
    def __init__(self, question):  
        """Store a question, and prepare to store responses."""  
        self.question = question  
        self.responses = []
```
- ❷

```
    def show_question(self):  
        """Show the survey question."""  
        print(self.question)
```
- ❸

```
    def store_response(self, new_response):  
        """Store a single response to the survey."""  
        self.responses.append(new_response)
```
- ❹

```
    def show_results(self):  
        """Show all the responses that have been given."""  
        print("Survey results:")  
        for response in self.responses:  
            print(f"- {response}")
```

Esta clase comienza con una pregunta de encuesta que proporcionas a ❶ e incluye una lista vacía para almacenar las respuestas. La clase tiene métodos para imprimir la pregunta de la encuesta ❷ , añadir una nueva respuesta a la lista de respuestas ❸ , e imprimir todas las respuestas almacenadas en la lista ❹ . Para crear una instancia a partir de esta clase, todo lo que tienes que proporcionar es una pregunta. Una vez que tengas una instancia que represente una encuesta concreta, muestra la pregunta de la encuesta con `show_question()` , almacena una respuesta utilizando `store_response()` , y muestra los resultados con `show_results()` .

This class starts with a survey question that you provide ❶ and includes an empty list to store responses. The class has methods to print the survey question ❷ , add a new response to the response list ❸ , and print all the responses stored in the list ❹ . To create an instance from this class, all you have to provide is a question. Once you have an instance representing a particular survey, you display the survey question with `show_question()` , store a response using `store_response()` , and show results with `show_results()` .

Para demostrar que la clase `AnonymousSurvey` funciona, vamos a escribir un programa que utilice la clase:

To show that the `AnonymousSurvey` class works, let's write a program that uses the class:

lenguaje_encuesta.py

language_survey.py

```
from survey import AnonymousSurvey

# Define a question, and make a survey.
question = "What language did you first learn to speak?"
language_survey = AnonymousSurvey(question)

# Show the question, and store responses to the question.
language_survey.show_question()
print("Enter 'q' at any time to quit.\n")
while True:
    response = input("Language: ")
    if response == 'q':
        break
    language_survey.store_response(response)

# Show the survey results.
print("\nThank you to everyone who participated in the survey!")
language_survey.show_results()
```

Este programa define una pregunta ("What language did you first learn to speak?") y crea un objeto `AnonymousSurvey` con esa pregunta. El programa llama a `show_question()` para que muestre la pregunta y luego pide respuestas. Cada respuesta se almacena a medida que se recibe. Cuando se han introducido todas las respuestas (el usuario introduce `q` para salir), `show_results()` imprime los resultados de la encuesta:

This program defines a question ("What language did you first learn to speak?") and creates an `AnonymousSurvey` object with that question. The program calls `show_question()` to display the question and then prompts for responses. Each response is stored as it is received. When all responses have been entered (the user inputs `q` to quit), `show_results()` prints the survey results:

```
What language did you first learn to speak?  
Enter 'q' at any time to quit.
```

```
Language: English  
Language: Spanish  
Language: English  
Language: Mandarin  
Language: q
```

```
Thank you to everyone who participated in the survey!  
Survey results:
```

```
- English  
- Spanish  
- English  
- Mandarin
```

Esta clase funciona para una simple encuesta anónima, pero supongamos que queremos mejorar `AnonymousSurvey` y el módulo en el que se encuentra, `survey` . Podríamos permitir que cada usuario introdujera más de una respuesta, podríamos escribir un método para listar sólo las respuestas únicas e informar de cuántas veces se ha dado cada respuesta, o incluso podríamos escribir otra clase para gestionar las encuestas no anónimas.

This class works for a simple anonymous survey, but say we want to improve `AnonymousSurvey` and the module it's in, `survey` . We could allow each user to enter more than one response, we could write a method to list only unique responses and to report how many times each response was given, or we could even write another class to manage non-anonymous surveys.

Implementar tales cambios podría afectar al comportamiento actual de la clase `AnonymousSurvey`. Por ejemplo, es posible que, al intentar permitir que cada usuario introduzca varias respuestas, cambiemos accidentalmente el modo en que se gestionan las respuestas únicas. Para asegurarnos de que no rompemos el comportamiento existente mientras desarrollamos este módulo, podemos escribir pruebas para la clase.

Implementing such changes would risk affecting the current behavior of the class `AnonymousSurvey`. For example, it's possible that while trying to allow each user to enter multiple responses, we could accidentally change how single responses are handled. To ensure we don't break existing behavior as we develop this module, we can write tests for the class.

Prueba de la clase `EncuestaAnónima` (*Testing the `AnonymousSurvey` Class*)

Vamos a escribir una prueba que verifique un aspecto del comportamiento de `AnonymousSurvey`. Escribiremos una prueba para verificar que una única respuesta a la pregunta de la encuesta se almacena correctamente:

Let's write a test that verifies one aspect of the way `AnonymousSurvey` behaves. We'll write a test to verify that a single response to the survey question is stored properly:

prueba_encuesta.py

test_survey.py

```
from survey import AnonymousSurvey
```

```
❶ def test_store_single_response():
    """Test that a single response is stored properly."""
    question = "What language did you first learn to speak?"
    ❷ language_survey = AnonymousSurvey(question)
    language_survey.store_response('English')
    ❸ assert 'English' in language_survey.responses
```

Empezaremos importando la clase que queremos probar, `AnonymousSurvey` . La primera función de prueba verificará que cuando almacenemos una respuesta a la pregunta de la encuesta, la respuesta acabará en la lista de respuestas de la encuesta. Un buen nombre descriptivo para esta función es `test_store_single_response()` ❶ . Si esta prueba falla, sabremos por el nombre de la función en el resumen de la prueba que hubo un problema al almacenar una única respuesta a la encuesta.

We start by importing the class we want to test, `AnonymousSurvey` . The first test function will verify that when we store a response to the survey question, the response will end up in the survey's list of responses. A good descriptive name for this function is `test_store_single_response()` ❶ . If this test fails, we'll know from the function name in the test summary that there was a problem storing a single response to the survey.

Para probar el comportamiento de una clase, necesitamos crear una instancia de la clase. Creamos una instancia llamada `language_survey` ❷ con la pregunta "What language did you first learn to speak?" Almacenamos una única respuesta, `English` , utilizando el método `store_response()` . A continuación, verificamos que la respuesta se ha almacenado correctamente afirmando que `English` está en la lista `language_survey.responses` ❸ .

To test the behavior of a class, we need to make an instance of the class. We create an instance called `language_survey` ❷ with the question "What language did you first learn to speak?" We store a single response, `English` , using the `store_response()` method. Then we verify that the response was stored correctly by asserting that `English` is in the list `language_survey.responses` ❸ .

Por defecto, al ejecutar el comando `pytest` sin argumentos, se ejecutarán todas las pruebas que `pytest` descubra en el directorio actual. Para centrarte en las pruebas de un archivo, pasa el nombre del archivo de pruebas que quieras ejecutar. Aquí ejecutaremos sólo la prueba que escribimos para `AnonymousSurvey` :

By default, running the command `pytest` with no arguments will run all the tests that `pytest` discovers in the current directory. To focus on the tests in one file, pass the name of the test file you want to run. Here we'll run just the one test we wrote for `AnonymousSurvey` :

```
$ pytest test_survey.py
===== test session starts =====
--snip--
test_survey.py . [100%]
===== 1 passed in 0.01s =====
```

Es un buen comienzo, pero una encuesta sólo es útil si genera más de una respuesta. Verifiquemos que se pueden almacenar tres respuestas correctamente. Para ello, añadimos otro método a `TestAnonymousSurvey` :

This is a good start, but a survey is useful only if it generates more than one response. Let's verify that three responses can be stored correctly. To do this, we add another method to `TestAnonymousSurvey` :

```
from survey import AnonymousSurvey

def test_store_single_response():
    --snip--

def test_store_three_responses():
    """Test that three individual responses are stored properly."""
    question = "What language did you first learn to speak?"
    language_survey = AnonymousSurvey(question)
    ❶ responses = ['English', 'Spanish', 'Mandarin']
    for response in responses:
        language_survey.store_response(response)

    ❷ for response in responses:
        assert response in language_survey.responses
```

Llamamos a la nueva función `test_store_three_responses()` . Creamos un objeto encuesta igual que hicimos en `test_store_single_response()` . Definimos una lista que contenga tres respuestas diferentes ❶ , y luego llamamos a `store_response()` para cada una de estas respuestas. Una vez almacenadas las respuestas, escribimos otro bucle y afirmamos que cada respuesta está ahora en `language_survey.responses` ❷ .

We call the new function `test_store_three_responses()` . We create a survey object just like we did in `test_store_single_response()` . We define a list containing three different responses ❶ , and then we call `store_response()` for each of these responses. Once the

responses have been stored, we write another loop and assert that each response is now in `language_survey.responses` ② .

Cuando volvemos a ejecutar el archivo de prueba, ambas pruebas (para una sola respuesta y para tres respuestas) pasan:

When we run the test file again, both tests (for a single response and for three responses) pass:

```
$ pytest test_survey.py
===== test session starts =====
--snip--
test_survey.py .. [100%]
===== 2 passed in 0.01s =====
```

Esto funciona perfectamente. Sin embargo, estas pruebas son un poco repetitivas, así que utilizaremos otra función de `pytest` para hacerlas más eficientes.

This works perfectly. However, these tests are a bit repetitive, so we'll use another feature of `pytest` to make them more efficient.

Utilizar dispositivos (Using Fixtures)

En `test_survey.py`, creamos una nueva instancia de `AnonymousSurvey` en cada función de prueba. Esto está bien en el breve ejemplo con el que estamos trabajando, pero en un proyecto del mundo real con decenas o cientos de pruebas, esto sería problemático.

In `test_survey.py`, we created a new instance of `AnonymousSurvey` in each test function. This is fine in the short example we're working with, but in a real-world project with tens or hundreds of tests, this would be problematic.

En las pruebas, un *fixture* ayuda a configurar un entorno de pruebas. A menudo, esto significa crear un recurso que es utilizado por más de una prueba. Creamos un fixture en `pytest` escribiendo una función con el decorador `@pytest.fixture`. Un *decorator* es una directiva que se coloca justo antes de la definición de una función; Python aplica esta

directiva a la función antes de que se ejecute, para alterar cómo se comporta el código de la función. No te preocupes si esto te parece complicado; puedes empezar a utilizar decoradores de paquetes de terceros antes de aprender a escribirlos tú mismo.

In testing, a *fixture* helps set up a test environment. Often, this means creating a resource that's used by more than one test. We create a fixture in `pytest` by writing a function with the decorator `@pytest.fixture`. A *decorator* is a directive placed just before a function definition; Python applies this directive to the function before it runs, to alter how the function code behaves. Don't worry if this sounds complicated; you can start to use decorators from third-party packages before learning to write them yourself.

Utilicemos un fixture para crear una única instancia de encuesta que pueda utilizarse en ambas funciones de prueba en `test_survey.py`:

Let's use a fixture to create a single survey instance that can be used in both test functions in `test_survey.py`:

```
import pytest
from survey import AnonymousSurvey

❶ @pytest.fixture
❷ def language_survey():
    """A survey that will be available to all test functions."""
    question = "What language did you first learn to speak?"
    language_survey = AnonymousSurvey(question)
    return language_survey

❸ def test_store_single_response(language_survey):
    """Test that a single response is stored properly."""
❹     language_survey.store_response('English')
    assert 'English' in language_survey.responses

❺ def test_store_three_responses(language_survey):
    """Test that three individual responses are stored properly."""
    responses = ['English', 'Spanish', 'Mandarin']
    for response in responses:
❻         language_survey.store_response(response)
```

```
for response in responses:
    assert response in language_survey.responses
```

Ahora necesitamos importar `pytest` , porque estamos utilizando un decorador que está definido en `pytest` . Aplicamos el decorador `@pytest.fixture` ❶ a la nueva función `language_survey()` ❷ . Esta función construye un objeto `AnonymousSurvey` y devuelve la nueva encuesta.

We need to import `pytest` now, because we're using a decorator that's defined in `pytest` . We apply the `@pytest.fixture` decorator ❶ to the new function `language_survey()` ❷ . This function builds an `AnonymousSurvey` object and returns the new survey.

Observa que las definiciones de ambas funciones de prueba han cambiado ❸ ❺ ; cada función de prueba tiene ahora un parámetro llamado `language_survey` . Cuando un parámetro de una función de prueba coincide con el nombre de una función con el decorador `@pytest.fixture` , el accesorio se ejecutará automáticamente y el valor devuelto se pasará a la función de prueba. En este ejemplo, la función `language_survey()` suministra tanto a `test_store_single_response()` como a `test_store_three_responses()` una instancia de `language_survey` .

Notice that the definitions of both test functions have changed ❸ ❺ ; each test function now has a parameter called `language_survey` . When a parameter in a test function matches the name of a function with the `@pytest.fixture` decorator, the fixture will be run automatically and the return value will be passed to the test function. In this example, the function `language_survey()` supplies both `test_store_single_response()` and `test_store_three_responses()` with a `language_survey` instance.

No hay código nuevo en ninguna de las funciones de prueba, pero observa que se han eliminado dos líneas de cada función ❹ ❻ : la línea que definía una pregunta y la línea que creaba un objeto `AnonymousSurvey` .

There's no new code in either of the test functions, but notice that two lines have been removed from each function ❹ ❻ : the line that defined a question and the line that created an `AnonymousSurvey` object.

Cuando volvemos a ejecutar el archivo de prueba, ambas pruebas siguen pasando. Estas pruebas serían especialmente útiles cuando intentáramos ampliar `AnonymousSurvey` para manejar múltiples respuestas para cada persona. Después de modificar el código para que acepte múltiples respuestas, podrías ejecutar estas pruebas y asegurarte de que no has afectado a la capacidad de almacenar una sola respuesta o una serie de respuestas individuales.

When we run the test file again, both tests still pass. These tests would be particularly useful when trying to expand `AnonymousSurvey` to handle multiple responses for each person. After modifying the code to accept multiple responses, you could run these tests and make sure you haven't affected the ability to store a single response or a series of individual responses.

Es casi seguro que la estructura anterior te parecerá complicada; contiene parte del código más abstracto que has visto hasta ahora. No es necesario que utilices fixtures de inmediato; es mejor escribir pruebas que tengan mucho código repetitivo que no escribir ninguna prueba. Sólo tienes que saber que cuando hayas escrito suficientes pruebas como para que la repetición sea un estorbo, existe una forma bien establecida de tratar la repetición. Además, en ejemplos sencillos como éste, las fijaciones no hacen que el código sea más corto ni más fácil de seguir. Pero en proyectos con muchas pruebas, o en situaciones en las que se necesitan muchas líneas para construir un recurso que se utiliza en varias pruebas, las fijaciones pueden mejorar drásticamente tu código de prueba.

The structure above will almost certainly look complicated; it contains some of the most abstract code you've seen so far. You don't need to use fixtures right away; it's better to write tests that have a lot of repetitive code than to write no tests at all. Just know that when you've written enough tests that the repetition is getting in the way, there's a well-established way to deal with the repetition. Also, fixtures in simple examples like this one don't really make the code any shorter or simpler to follow. But in projects with many tests, or in situations where it takes many lines to build a resource that's used in multiple tests, fixtures can drastically improve your test code.

Cuando quieras escribir un fixture, escribe una función que genere el recurso que utilizan varias funciones de prueba. Añade el decorador `@pytest.fixture` a la nueva función, y añade el nombre de esta función como parámetro para cada función de prueba que utilice este recurso. A partir de ese momento, tus pruebas serán más cortas y más fáciles de escribir y mantener.

When you want to write a fixture, write a function that generates the resource that's used by multiple test functions. Add the `@pytest.fixture` decorator to the new function, and add the name of this function as a parameter for each test function that uses this resource. Your tests will be shorter and easier to write and maintain from that point forward.

Pruébalo tú mismo (*Try It Yourself*)

11-3. Empleado: Escribe una clase llamada `Employee`. El método `__init__()` debe tomar un nombre, un apellido y un salario anual, y almacenar cada uno de ellos como atributos. Escribe un método llamado `give_raise()` que añada 5.000\$ al salario anual por defecto, pero que también acepte una cantidad de aumento diferente.

11-3. Employee: Write a class called `Employee`. The `__init__()` method should take in a first name, a last name, and an annual salary, and store each of these as attributes. Write a method called `give_raise()` that adds \$5,000 to the annual salary by default but also accepts a different raise amount.

Escribe un archivo de prueba para `Employee` con dos funciones de prueba, `test_give_default_raise()` y `test_give_custom_raise()`. Escribe tus pruebas una vez sin utilizar un fixture, y asegúrate de que ambas pasan. A continuación, escribe un fixture para no tener que crear una nueva instancia de empleado en cada función de prueba. Vuelve a ejecutar las pruebas y asegúrate de que ambas siguen siendo correctas.

Write a test file for `Employee` with two test functions, `test_give_default_raise()` and `test_give_custom_raise()`. Write your tests once without using a fixture, and make sure they both pass. Then write a fixture so you don't have to create a new employee instance in each test function. Run the tests again, and make sure both tests still pass.

Resumen (*Summary*)

En este capítulo has aprendido a escribir pruebas para funciones y clases utilizando las herramientas del módulo `pytest`. Has aprendido a escribir funciones de prueba que verifican comportamientos específicos que deben mostrar tus funciones y clases. Has visto cómo se pueden utilizar las fijaciones para crear eficientemente recursos que se pueden

utilizar en múltiples funciones de prueba en un archivo de prueba.

In this chapter, you learned to write tests for functions and classes using tools in the `pytest` module. You learned to write test functions that verify specific behaviors your functions and classes should exhibit. You saw how fixtures can be used to efficiently create resources that can be used in multiple test functions in a test file.

Las pruebas son un tema importante al que muchos programadores noveles no están expuestos. No tienes que escribir pruebas para todos los proyectos sencillos que pruebes como nuevo programador. Pero en cuanto empieces a trabajar en proyectos que impliquen un esfuerzo de desarrollo significativo, debes probar los comportamientos críticos de tus funciones y clases. Estarás más seguro de que el nuevo trabajo en tu proyecto no romperá las partes que funcionan, y esto te dará libertad para introducir mejoras en tu código. Si accidentalmente rompes una funcionalidad existente, lo sabrás enseguida, por lo que podrás solucionar el problema fácilmente. Responder a una prueba fallida que has ejecutado es mucho más fácil que responder a un informe de error de un usuario descontento.

Testing is an important topic that many newer programmers aren't exposed to. You don't have to write tests for all the simple projects you try as a new programmer. But as soon as you start to work on projects that involve significant development effort, you should test the critical behaviors of your functions and classes. You'll be more confident that new work on your project won't break the parts that work, and this will give you the freedom to make improvements to your code. If you accidentally break existing functionality, you'll know right away, so you can still fix the problem easily. Responding to a failed test that you ran is much easier than responding to a bug report from an unhappy user.

Otros programadores respetarán más tus proyectos si incluyes algunas pruebas iniciales. Se sentirán más cómodos experimentando con tu código y estarán más dispuestos a trabajar contigo en los proyectos. Si quieres contribuir a un proyecto en el que están trabajando otros programadores, se esperará de ti que demuestres que tu código supera las pruebas existentes y, normalmente, se esperará de ti que escribas pruebas para cualquier comportamiento nuevo que introduzcas en el proyecto.

Other programmers will respect your projects more if you include some initial tests. They'll feel more comfortable experimenting with your code and be more willing to work with you on projects. If you want to contribute to a project that other programmers are working on,

you'll be expected to show that your code passes existing tests and you'll usually be expected to write tests for any new behavior you introduce to the project.

Juega con las pruebas para familiarizarte con el proceso de comprobación de tu código. Escribe pruebas para los comportamientos más críticos de tus funciones y clases, pero no pretendas una cobertura total en los primeros proyectos a menos que tengas una razón específica para hacerlo.

Play around with tests to become familiar with the process of testing your code. Write tests for the most critical behaviors of your functions and classes, but don't aim for full coverage in early projects unless you have a specific reason to do so.

Parte II: Proyectos (Part II: Projects)

Parte II Proyectos (*Part II Projects*)

¡Enhorabuena! Ya sabes lo suficiente sobre Python como para empezar a crear proyectos interactivos y significativos. Crear tus propios proyectos te enseñará nuevas habilidades y consolidará tu comprensión de los conceptos introducidos en la Parte I.

Congratulations! You now know enough about Python to start building interactive and meaningful projects. Creating your own projects will teach you new skills and solidify your understanding of the concepts introduced in Part I.

La Parte II contiene tres tipos de proyectos, y puedes elegir realizar cualquiera de ellos o todos en el orden que desees. He aquí una breve descripción de cada proyecto para ayudarte a decidir en cuál profundizar primero.

Part II contains three kinds of projects, and you can choose to do any or all of these projects in whichever order you like. Here's a brief description of each project to help you decide which to dig into first.

Invasión alienígena: Crear un juego con Python (***Alien Invasion: Making a Game with Python***)

En el proyecto Invasión alienígena (**Capítulos 12 , 13 , y 14**), utilizarás el paquete Pygame para desarrollar un juego en 2D. El objetivo del juego es derribar una flota de alienígenas a medida que caen por la pantalla, en niveles que aumentan en velocidad y dificultad. Al final del proyecto, habrás aprendido habilidades que te permitirán desarrollar tus propios juegos 2D en Pygame.

In the Alien Invasion project (**Chapters 12 , 13 , and 14**), you'll use the Pygame package to develop a 2D game. The goal of the game is to shoot down a fleet of aliens as they drop down the screen, in levels that increase in speed and difficulty. At the end of the project, you'll have learned skills that will enable you to develop your own 2D games in Pygame.

Visualización de datos (*Data Visualization*)

Los proyectos de Visualización de Datos comienzan en el **Capítulo 15**, donde aprenderás a generar datos y a crear una serie de visualizaciones funcionales y bonitas de esos datos utilizando Matplotlib y Plotly. **El Capítulo 16** te enseña a acceder a datos de fuentes en línea y a introducirlos en un paquete de visualización para crear gráficos de datos meteorológicos y un mapa de la actividad sísmica mundial. Por último, **el Capítulo 17** te muestra cómo escribir un programa para descargar automáticamente y visualizar datos. Aprender a hacer visualizaciones te permite explorar el campo de la ciencia de datos, que es una de las áreas de la programación con mayor demanda en la actualidad.

The Data Visualization projects start in **Chapter 15**, where you'll learn to generate data and create a series of functional and beautiful visualizations of that data using Matplotlib and Plotly. **Chapter 16** teaches you to access data from online sources and feed it into a visualization package to create plots of weather data and a map of global earthquake activity. Finally, **Chapter 17** shows you how to write a program to automatically download and visualize data. Learning to make visualizations allows you to explore the field of data science, which is one of the highest-demand areas of programming today.

Aplicaciones Web (*Web Applications*)

En el proyecto Aplicación Web (**Capítulos 18, 19, y 20**), utilizarás el paquete Django para crear una sencilla aplicación web que permita a los usuarios llevar un diario sobre los distintos temas que han ido aprendiendo. Los usuarios crearán una cuenta con un nombre de usuario y una contraseña, introducirán un tema y luego harán entradas sobre lo que están aprendiendo. También desplegarás tu aplicación en un servidor remoto para que cualquier persona del mundo pueda acceder a ella.

In the Web Application project (**Chapters 18, 19, and 20**), you'll use the Django package to create a simple web application that allows users to keep a journal about different topics they've been learning about. Users will create an account with a username and password, enter a topic, and then make entries about what they're learning. You'll also deploy your app to a remote server so anyone in the world can access it.

Tras completar este proyecto, podrás empezar a crear tus propias aplicaciones web sencillas, y estarás preparado para profundizar en recursos más completos sobre la creación de aplicaciones con Django.

After completing this project, you'll be able to start building your own simple web applications, and you'll be ready to delve into more thorough resources on building applications with Django.

Capítulo 12: Un barco que dispara balas (Chapter 12: A Ship That Fires Bullets)

12

Un barco que dispara balas (12 A Ship That Fires Bullets)



DESCRIPCIÓN DE LA IMAGEN (En español):

La imagen muestra una ilustración en blanco y negro de una serpiente con gafas. La serpiente está enrollada alrededor de un motor a reacción, el cual parece estar listo para despegar. La expresión de la serpiente es decidida y confiada. El diseño tiene un estilo caricaturesco y dinámico, destacando el movimiento y la fuerza del motor. El fondo es de un tono gris claro que resalta el dibujo principal.

¡Vamos a construir un juego llamado *Alien Invasion* ! Utilizaremos Pygame, una colección de divertidos y potentes módulos de Python que gestionan los gráficos, la animación e incluso el sonido, facilitándote la construcción de juegos sofisticados. Con Pygame encargándose de tareas como dibujar imágenes en la pantalla, puedes centrarte en la lógica de alto nivel de la dinámica del juego.

Let's build a game called *Alien Invasion* ! We'll use Pygame, a collection of fun, powerful Python modules that manage graphics, animation, and even sound, making it easier for you to build sophisticated games. With Pygame handling tasks like drawing images to the screen, you can focus on the higher-level logic of game dynamics.

En este capítulo, configurarás Pygame y crearás un cohete espacial que se mueve a derecha e izquierda y dispara balas en respuesta a las entradas del jugador. En los dos capítulos

siguientes, crearás una flota de alienígenas para destruir, y luego seguirás perfeccionando el juego estableciendo límites al número de naves que puedes utilizar y añadiendo un marcador.

In this chapter, you'll set up Pygame and then create a rocket ship that moves right and left and fires bullets in response to player input. In the next two chapters, you'll create a fleet of aliens to destroy, and then continue to refine the game by setting limits on the number of ships you can use and adding a scoreboard.

Mientras construyes este juego, también aprenderás a gestionar grandes proyectos que abarcan varios archivos. Refactorizaremos mucho código y gestionaremos el contenido de los archivos para organizar el proyecto y hacer que el código sea eficiente.

While building this game, you'll also learn how to manage large projects that span multiple files. We'll refactor a lot of code and manage file contents to organize the project and make the code efficient.

Hacer juegos es una forma ideal de divertirse mientras se aprende un lenguaje. Es profundamente satisfactorio jugar a un juego que has escrito tú, y escribir un juego sencillo te enseñará mucho sobre cómo desarrollan juegos los profesionales. Mientras trabajas en este capítulo, introduce y ejecuta el código para identificar cómo contribuye cada bloque de código a la jugabilidad general. Experimenta con distintos valores y ajustes para comprender mejor cómo perfeccionar las interacciones en tus juegos.

Making games is an ideal way to have fun while learning a language. It's deeply satisfying to play a game you wrote, and writing a simple game will teach you a lot about how professionals develop games. As you work through this chapter, enter and run the code to identify how each code block contributes to overall gameplay. Experiment with different values and settings to better understand how to refine interactions in your games.

Nota (Note)

Alien Invasion abarca varios archivos diferentes, así que crea una nueva carpeta *alien_invasion* en tu sistema. Asegúrate de guardar todos los archivos del proyecto en esta carpeta para que tus sentencias `import` funcionen correctamente.

Alien Invasion spans a number of different files, so make a new *alien_invasion* folder on your system. Be sure to save all files for the project to this folder so your `import` statements will work correctly.

Además, si te sientes cómodo utilizando el control de versiones, quizás quieras utilizarlo para este proyecto. Si no has utilizado antes el control de versiones, consulta el Apéndice D para obtener una visión general.

Also, if you feel comfortable using version control, you might want to use it for this project. If you haven't used version control before, see Appendix D for an overview.

Planificar tu proyecto (*Planning Your Project*)

Cuando estás construyendo un gran proyecto, es importante preparar un plan antes de empezar a escribir código. Tu plan te mantendrá centrado y hará más probable que completes el proyecto.

When you're building a large project, it's important to prepare a plan before you begin to write code. Your plan will keep you focused and make it more likely that you'll complete the project.

Escribamos una descripción de la jugabilidad general. Aunque la siguiente descripción no cubre todos los detalles de *Alien Invasion*, proporciona una idea clara de cómo empezar a construir el juego:

Let's write a description of the general gameplay. Although the following description doesn't cover every detail of *Alien Invasion*, it provides a clear idea of how to start building the game:

En *Alien Invasion*, el jugador controla una nave cohete que aparece en la parte inferior central de la pantalla. El jugador puede mover la nave a derecha e

izquierda con las teclas de flecha y disparar balas con la barra espaciadora. Cuando comienza el juego, una flota de alienígenas llena el cielo y se desplaza por la pantalla. El jugador dispara y destruye a los alienígenas. Si el jugador destruye a todos los alienígenas, aparece una nueva flota que se mueve más rápido que la anterior. Si algún alienígena golpea la nave del jugador o llega al fondo de la pantalla, el jugador pierde una nave. Si el jugador pierde tres naves, el juego termina.

In *Alien Invasion*, the player controls a rocket ship that appears at the bottom center of the screen. The player can move the ship right and left using the arrow keys and shoot bullets using the spacebar. When the game begins, a fleet of aliens fills the sky and moves across and down the screen. The player shoots and destroys the aliens. If the player destroys all the aliens, a new fleet appears that moves faster than the previous fleet. If any alien hits the player's ship or reaches the bottom of the screen, the player loses a ship. If the player loses three ships, the game ends.

Para la primera fase de desarrollo, haremos una nave que pueda moverse a derecha e izquierda cuando el jugador pulse las teclas de flecha y disparar balas cuando el jugador pulse la barra espaciadora. Después de configurar este comportamiento, podemos crear los alienígenas y perfeccionar la jugabilidad.

For the first development phase, we'll make a ship that can move right and left when the player presses the arrow keys and fire bullets when the player presses the spacebar. After setting up this behavior, we can create the aliens and refine the gameplay.

Instalación de Pygame (*Installing Pygame*)

Antes de empezar a programar, instala Pygame. Lo haremos del mismo modo que instalamos pytest en el Capítulo 11 : con pip. Si te has saltado el Capítulo 11 o necesitas un repaso sobre pip, consulta " Instalar pytest con pip " en la página 210 .

Before you begin coding, install Pygame. We'll do this the same way we installed pytest in Chapter 11 : with pip. If you skipped Chapter 11 or need a refresher on pip, see " Installing pytest with pip " on page 210 .

Para instalar Pygame, introduce el siguiente comando en una terminal:

To install Pygame, enter the following command at a terminal prompt:

```
$ python -m pip install --user pygame
```

Si utilizas un comando distinto de `python` para ejecutar programas o iniciar una sesión de terminal, como `python3`, asegúrate de utilizar ese comando en su lugar.

If you use a command other than `python` to run programs or start a terminal session, such as `python3`, make sure you use that command instead.

Iniciar el proyecto de juego (*Starting the Game Project*)

Empezaremos a construir el juego creando una ventana Pygame vacía. Más tarde, dibujaremos los elementos del juego, como la nave y los alienígenas, en esta ventana. También haremos que nuestro juego responda a las entradas del usuario, estableceremos el color de fondo y cargaremos una imagen de la nave.

We'll begin building the game by creating an empty Pygame window. Later, we'll draw the game elements, such as the ship and the aliens, on this window. We'll also make our game respond to user input, set the background color, and load a ship image.

Crear una ventana Pygame y responder a las entradas del usuario (*Creating a Pygame Window and Responding to User Input*)

Crearemos una ventana Pygame vacía creando una clase que represente el juego. En tu editor de texto, crea un nuevo archivo y guárdalo como *alien_invasion.py*; luego introduce lo siguiente:

We'll make an empty Pygame window by creating a class to represent the game. In your text editor, create a new file and save it as *alien_invasion.py*; then enter the following:

alien_invasion.py

alien_invasion.py

```
import sys

import pygame

class AlienInvasion:
    """Overall class to manage game assets and behavior."""

    def __init__(self):
        """Initialize the game, and create game resources."""
        ❶ pygame.init()

        ❷ self.screen = pygame.display.set_mode((1200, 800))
        pygame.display.set_caption("Alien Invasion")

    def run_game(self):
        """Start the main loop for the game."""
        ❸ while True:
            # Watch for keyboard and mouse events.
            ❹ for event in pygame.event.get():
                ❺ if event.type == pygame.QUIT:
                    sys.exit()

            # Make the most recently drawn screen visible.
            ❻ pygame.display.flip()

if __name__ == '__main__':
    # Make a game instance, and run the game.
    ai = AlienInvasion()
    ai.run_game()
```

En primer lugar, importamos los módulos `sys` y `pygame`. El módulo `pygame` contiene la funcionalidad que necesitamos para hacer un juego. Utilizaremos las herramientas del módulo `sys` para salir del juego cuando el jugador lo abandone.

First, we import the `sys` and `pygame` modules. The `pygame` module contains the functionality we need to make a game. We'll use tools in the `sys` module to exit the game when the player quits.

Alien Invasion comienza como una clase llamada `AlienInvasion`. En el método `__init__()`, la función `pygame.init()` inicializa los ajustes de fondo que Pygame necesita para funcionar correctamente ❶. Luego llamamos a `pygame.display.set_mode()` para crear una ventana de visualización ❷, en la que dibujaremos todos los elementos gráficos del juego. El argumento `(1200, 800)` es una tupla que define las dimensiones de la ventana del juego, que será de 1.200 píxeles de ancho por 800 píxeles de alto. (Puedes ajustar estos valores en función del tamaño de tu pantalla.) Asignamos esta ventana de visualización al atributo `self.screen`, para que esté disponible en todos los métodos de la clase.

Alien Invasion starts as a class called `AlienInvasion`. In the `__init__()` method, the `pygame.init()` function initializes the background settings that Pygame needs to work properly ❶. Then we call `pygame.display.set_mode()` to create a display window ❷, on which we'll draw all the game's graphical elements. The argument `(1200, 800)` is a tuple that defines the dimensions of the game window, which will be 1,200 pixels wide by 800 pixels high. (You can adjust these values depending on your display size.) We assign this display window to the attribute `self.screen`, so it will be available in all methods in the class.

El objeto que asignamos a `self.screen` se llama superficie. Una *surface* en Pygame es una parte de la pantalla donde se puede mostrar un elemento del juego. Cada elemento del juego, como un alienígena o una nave, es su propia superficie. La superficie devuelta por `display.set_mode()` representa toda la ventana del juego. Cuando activemos el bucle de animación del juego, esta superficie se redibujará en cada pasada por el bucle, para que pueda actualizarse con cualquier cambio provocado por la entrada del usuario.

The object we assigned to `self.screen` is called a surface. A *surface* in Pygame is a part of the screen where a game element can be displayed. Each element in the game, like an alien or a ship, is its own surface. The surface returned by `display.set_mode()` represents the entire game window. When we activate the game's animation loop, this surface will be redrawn on every pass through the loop, so it can be updated with any changes triggered by user input.

El juego se controla mediante el método `run_game()` . Este método contiene un bucle `while` ❸ que se ejecuta continuamente. El bucle `while` contiene un bucle de eventos y código que gestiona las actualizaciones de la pantalla. Un *event* es una acción que el usuario realiza mientras juega, como pulsar una tecla o mover el ratón. Para que nuestro programa responda a los eventos, escribimos un bucle *event loop* a *listen* para los eventos y realizar las tareas adecuadas en función del tipo de eventos que se produzcan. El bucle `for` ❹ anidado dentro del bucle `while` es un bucle de eventos.

The game is controlled by the `run_game()` method. This method contains a `while` loop ❸ that runs continually. The `while` loop contains an event loop and code that manages screen updates. An *event* is an action that the user performs while playing the game, such as pressing a key or moving the mouse. To make our program respond to events, we write an *event loop* to *listen* for events and perform appropriate tasks depending on the kinds of events that occur. The `for` loop ❹ nested inside the `while` loop is an event loop.

Para acceder a los eventos que Pygame detecta, utilizaremos la función `pygame.event.get()` . Esta función devuelve una lista de eventos que han tenido lugar desde la última vez que se llamó a esta función. Cualquier evento de teclado o ratón hará que se ejecute este bucle `for` . Dentro del bucle, escribiremos una serie de sentencias `if` para detectar y responder a eventos específicos. Por ejemplo, cuando el jugador haga clic en el botón de cierre de la ventana del juego, se detectará un evento `pygame.QUIT` y llamaremos a `sys.exit()` para salir del juego ❺ .

To access the events that Pygame detects, we'll use the `pygame.event.get()` function. This function returns a list of events that have taken place since the last time this function was called. Any keyboard or mouse event will cause this `for` loop to run. Inside the loop, we'll write a series of `if` statements to detect and respond to specific events. For example, when the player clicks the game window's close button, a `pygame.QUIT` event is detected and we call `sys.exit()` to exit the game ❺ .

La llamada a `pygame.display.flip()` ❻ indica a Pygame que haga visible la pantalla dibujada más recientemente. En este caso, simplemente dibuja una pantalla vacía en cada pasada por el bucle `while` , borrando la pantalla anterior para que sólo sea visible la nueva. Cuando movemos los elementos del juego, `pygame.display.flip()` actualiza continuamente la pantalla para mostrar las nuevas posiciones de los elementos del juego y ocultar las antiguas, creando la ilusión de un movimiento suave.

The call to `pygame.display.flip()` ❹ tells Pygame to make the most recently drawn screen visible. In this case, it simply draws an empty screen on each pass through the `while` loop, erasing the old screen so only the new screen is visible. When we move the game elements around, `pygame.display.flip()` continually updates the display to show the new positions of game elements and hide the old ones, creating the illusion of smooth movement.

Al final del archivo, creamos una instancia del juego y llamamos a `run_game()`. Colocamos `run_game()` en un bloque `if` que sólo se ejecuta si se llama directamente al archivo. Cuando ejecutes este archivo *alien_invasion.py*, deberías ver una ventana vacía de Pygame.

At the end of the file, we create an instance of the game and then call `run_game()`. We place `run_game()` in an `if` block that only runs if the file is called directly. When you run this *alien_invasion.py* file, you should see an empty Pygame window.

Control de la velocidad de fotogramas (*Controlling the Frame Rate*)

Idealmente, los juegos deberían ejecutarse a la misma velocidad, o *frame rate*, en todos los sistemas. Controlar la velocidad de fotogramas de un juego que puede ejecutarse en varios sistemas es una cuestión compleja, pero Pygame ofrece una forma relativamente sencilla de lograr este objetivo. Haremos un reloj, y nos aseguraremos de que el reloj haga tictac una vez en cada pasada por el bucle principal. Cada vez que el bucle procese más rápido que el ritmo que definamos, Pygame calculará el tiempo correcto de pausa para que el juego se ejecute a un ritmo constante.

Ideally, games should run at the same speed, or *frame rate*, on all systems. Controlling the frame rate of a game that can run on multiple systems is a complex issue, but Pygame offers a relatively simple way to accomplish this goal. We'll make a clock, and ensure the clock ticks once on each pass through the main loop. Anytime the loop processes faster than the rate we define, Pygame will calculate the correct amount of time to pause so that the game runs at a consistent rate.

Definiremos el reloj en el método `__init__()` :

We'll define the clock in the `__init__()` method:

alien_invasion.py

alien_invasion.py

```
def __init__(self):
    """Initialize the game, and create game resources."""
    pygame.init()
    self.clock = pygame.time.Clock()
    --snip--
```

Tras inicializar `pygame` , crearemos una instancia de la clase `Clock` , a partir del módulo `pygame.time` . Luego haremos que el reloj marque el final del bucle `while` en `run_game()` :

After initializing `pygame` , we create an instance of the class `Clock` , from the `pygame.time` module. Then we'll make the clock tick at the end of the `while` loop in `run_game()` :

```
def run_game(self):
    """Start the main loop for the game."""
    while True:
        --snip--
        pygame.display.flip()
        self.clock.tick(60)
```

El método `tick()` toma un argumento: la velocidad de fotogramas del juego. Aquí estoy utilizando un valor de 60, por lo que Pygame hará todo lo posible para que el bucle se ejecute exactamente 60 veces por segundo.

The `tick()` method takes one argument: the frame rate for the game. Here I'm using a value of 60, so Pygame will do its best to make the loop run exactly 60 times per second.

Nota (Note)

El reloj de Pygame debería ayudar a que el juego se ejecute de forma consistente en la mayoría de los sistemas. Si hace que el juego funcione de forma menos consistente en tu sistema, puedes probar con valores diferentes para la velocidad de fotogramas. Si no puedes encontrar una buena velocidad de fotogramas en tu sistema, puedes omitir por completo el reloj y ajustar la configuración del juego para que funcione bien en tu sistema.

Pygame's clock should help the game run consistently on most systems. If it makes the game run less consistently on your system, you can try different values for the frame rate. If you can't find a good frame rate on your system, you can leave the clock out entirely and adjust the game's settings so it runs well on your system.

Configurar el color de fondo (Setting the Background Color)

Pygame crea una pantalla negra por defecto, pero eso es aburrido. Vamos a establecer un color de fondo diferente. Lo haremos al final del método `__init__()`.

Pygame creates a black screen by default, but that's boring. Let's set a different background color. We'll do this at the end of the `__init__()` method.

invasión_extraterrestre.py

alien_invasion.py

```
def __init__(self):
    --snip--
    pygame.display.set_caption("Alien Invasion")

    # Set the background color.
    ❶ self.bg_color = (230, 230, 230)

def run_game(self):
    --snip--
    for event in pygame.event.get():
```

```

        if event.type == pygame.QUIT:
            sys.exit()

    ❷ # Redraw the screen during each pass through the loop.
        self.screen.fill(self.bg_color)

    # Make the most recently drawn screen visible.
    pygame.display.flip()
    self.clock.tick(60)

```

Los colores en Pygame se especifican como colores RGB: una mezcla de rojo, verde y azul. Cada valor de color puede ir de 0 a 255. El valor de color (255, 0, 0) es rojo, (0, 255, 0) es verde y (0, 0, 255) es azul. Puedes mezclar diferentes valores RGB para crear hasta 16 millones de colores. El valor de color (230, 230, 230) mezcla cantidades iguales de rojo, azul y verde, lo que produce un color de fondo gris claro. Asignamos este color a `self.bg_color` ❶ .

Colors in Pygame are specified as RGB colors: a mix of red, green, and blue. Each color value can range from 0 to 255. The color value (255, 0, 0) is red, (0, 255, 0) is green, and (0, 0, 255) is blue. You can mix different RGB values to create up to 16 million colors. The color value (230, 230, 230) mixes equal amounts of red, blue, and green, which produces a light gray background color. We assign this color to `self.bg_color` ❶ .

Rellenamos la pantalla con el color de fondo utilizando el método `fill()` ❷ , que actúa sobre una superficie y sólo toma un argumento: un color.

We fill the screen with the background color using the `fill()` method ❷ , which acts on a surface and takes only one argument: a color.

Creación de una clase de configuración (*Creating a Settings Class*)

Cada vez que introduzcamos una nueva funcionalidad en el juego, normalmente crearemos también algunos ajustes nuevos. En lugar de añadir ajustes por todo el código, escribamos un módulo llamado `settings` que contenga una clase llamada `Settings` para almacenar todos estos valores en un solo lugar. Este enfoque nos permite trabajar con un solo objeto `settings` cada vez que necesitemos acceder a un ajuste individual. Esto también facilita la modificación de la apariencia y el comportamiento del juego a medida que crece nuestro proyecto. Para modificar el juego, cambiaremos los valores relevantes en `settings.py` , que crearemos a continuación, en lugar de buscar diferentes ajustes por todo el proyecto.

Each time we introduce new functionality into the game, we'll typically create some new settings as well. Instead of adding settings throughout the code, let's write a module called `settings` that contains a class called `Settings` to store all these values in one place. This approach allows us to work with just one `settings` object anytime we need to access an individual setting. This also makes it easier to modify the game's appearance and behavior as our project grows. To modify the game, we'll change the relevant values in *settings.py*, which we'll create next, instead of searching for different settings throughout the project.

Crea un nuevo archivo llamado *settings.py* dentro de tu carpeta *alien_invasion*, y añade esta clase inicial `Settings` :

Create a new file named *settings.py* inside your *alien_invasion* folder, and add this initial `Settings` class:

ajustes.py

settings.py

```
class Settings:
    """A class to store all settings for Alien Invasion."""

    def __init__(self):
        """Initialize the game's settings."""
        # Screen settings
        self.screen_width = 1200
        self.screen_height = 800
        self.bg_color = (230, 230, 230)
```

Para crear una instancia de `Settings` en el proyecto y utilizarla para acceder a nuestra configuración, tenemos que modificar *alien_invasion.py* como se indica a continuación:

To make an instance of `Settings` in the project and use it to access our settings, we need to modify *alien_invasion.py* as follows:

alien_invasion.py

alien_invasion.py

```
--snip--
import pygame

from settings import Settings

class AlienInvasion:
    """Overall class to manage game assets and behavior."""

    def __init__(self):
        """Initialize the game, and create game resources."""
        pygame.init()
        self.clock = pygame.time.Clock()
        ❶ self.settings = Settings()

        ❷ self.screen = pygame.display.set_mode(
            (self.settings.screen_width, self.settings.screen_height))
        pygame.display.set_caption("Alien Invasion")

    def run_game(self):
        --snip--
        # Redraw the screen during each pass through the loop.
        ❸ self.screen.fill(self.settings.bg_color)

        # Make the most recently drawn screen visible.
        pygame.display.flip()
        self.clock.tick(60)
--snip--
```

Importamos `Settings` al archivo principal del programa. Luego creamos una instancia de `Settings` y la asignamos a `self.settings` ❶, después de hacer la llamada a `pygame.init()`. Cuando creamos una pantalla ❷, utilizamos los atributos `screen_width` y `screen_height` de `self.settings`, y luego utilizamos `self.settings` para acceder al color de fondo al rellenar la pantalla ❸ también.

We import `Settings` into the main program file. Then we create an instance of `Settings` and assign it to `self.settings` ❶, after making the call to `pygame.init()`. When we create a screen ❷, we use the `screen_width` and `screen_height` attributes of `self.settings`, and then we use `self.settings` to access the background color when filling the screen ❸ as well.

Cuando ejecutes ahora *alien_invasion.py* aún no verás ningún cambio, porque lo único que hemos hecho es trasladar a otro sitio los ajustes que ya estábamos utilizando. Ahora estamos listos para empezar a añadir nuevos elementos a la pantalla.

When you run *alien_invasion.py* now you won't yet see any changes, because all we've done is move the settings we were already using elsewhere. Now we're ready to start adding new elements to the screen.

Añadir la imagen de la nave (Adding the Ship Image)

Vamos a añadir la nave a nuestro juego. Para dibujar la nave del jugador en la pantalla, cargaremos una imagen y luego utilizaremos el método Pygame `blit()` para dibujar la imagen.

Let's add the ship to our game. To draw the player's ship on the screen, we'll load an image and then use the Pygame `blit()` method to draw the image.

Cuando elijas el material gráfico para tus juegos, asegúrate de prestar atención a las licencias. La forma más segura y barata de empezar es utilizar gráficos con licencia libre que puedas usar y modificar, de un sitio web como <https://opengameart.org>.

When you're choosing artwork for your games, be sure to pay attention to licensing. The safest and cheapest way to start is to use freely licensed graphics that you can use and modify, from a website like <https://opengameart.org>.

Puedes utilizar casi cualquier tipo de archivo de imagen en tu juego, pero es más fácil si utilizas un archivo de mapa de bits (*.bmp*) porque Pygame carga mapas de bits por defecto. Aunque puedes configurar Pygame para que utilice otros tipos de archivo, algunos tipos de archivo dependen de ciertas bibliotecas de imágenes que deben estar instaladas en

tu ordenador. La mayoría de las imágenes que encontrarás están en formato *.jpg* o *.png* , pero puedes convertirlas a mapas de bits utilizando herramientas como Photoshop, GIMP y Paint.

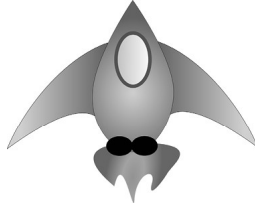
You can use almost any type of image file in your game, but it's easiest when you use a bitmap (*.bmp*) file because Pygame loads bitmaps by default. Although you can configure Pygame to use other file types, some file types depend on certain image libraries that must be installed on your computer. Most images you'll find are in *.jpg* or *.png* formats, but you can convert them to bitmaps using tools like Photoshop, GIMP, and Paint.

Presta especial atención al color de fondo de la imagen que elijas. Intenta encontrar un archivo con un fondo transparente o sólido que puedas sustituir por cualquier color de fondo, utilizando un editor de imágenes. Tus juegos tendrán mejor aspecto si el color de fondo de la imagen coincide con el color de fondo de tu juego. Alternativamente, puedes hacer coincidir el fondo de tu juego con el fondo de la imagen.

Pay particular attention to the background color in your chosen image. Try to find a file with a transparent or solid background that you can replace with any background color, using an image editor. Your games will look best if the image's background color matches your game's background color. Alternatively, you can match your game's background to the image's background.

Para *Alien Invasion* , puedes utilizar el archivo *ship.bmp* ([Figura 12-1](#)), que está disponible en los recursos de este libro en https://ehmatthes.github.io/pcc_3e . El color de fondo del archivo coincide con la configuración que estamos utilizando en este proyecto. Crea una carpeta llamada *images* dentro de tu carpeta principal del proyecto *alien_invasion* . Guarda el archivo *ship.bmp* en la carpeta *images* .

For *Alien Invasion* , you can use the file *ship.bmp* ([Figure 12-1](#)), which is available in this book's resources at https://ehmatthes.github.io/pcc_3e . The file's background color matches the settings we're using in this project. Make a folder called *images* inside your main *alien_invasion* project folder. Save the file *ship.bmp* in the *images* folder.



DESCRIPCIÓN DE LA IMAGEN (En español):

La imagen muestra un dibujo sencillo de un cohete espacial. Tiene una forma aerodinámica con un cuerpo principal en forma de gota invertida, con dos aletas a los lados que parecen alas. En la parte superior del cuerpo del cohete hay una forma ovalada que podría representar una ventana o una puerta. La base del cohete tiene dos círculos oscuros que simulan motores, y de ahí salen llamas estilizadas, sugiriendo movimiento o despegue. Todo el cohete está en tonos de gris, dando un aspecto metálico.

Figura 12-1 : La nave para Alien Invasion

Figure 12-1 : The ship for Alien Invasion

Crear la clase Nave (*Creating the Ship Class*)

Después de elegir una imagen para la nave, necesitamos mostrarla en la pantalla. Para utilizar nuestra nave, crearemos un nuevo módulo `ship` que contendrá la clase `Ship`. Esta clase gestionará la mayor parte del comportamiento de la nave del jugador:

After choosing an image for the ship, we need to display it on the screen. To use our ship, we'll create a new `ship` module that will contain the class `Ship`. This class will manage most of the behavior of the player's ship:

nave.py

ship.py

```
import pygame

class Ship:
    """A class to manage the ship."""

    def __init__(self, ai_game):
        """Initialize the ship and set its starting position."""
        ❶ self.screen = ai_game.screen
        ❷ self.screen_rect = ai_game.screen.get_rect()

        # Load the ship image and get its rect.
        ❸ self.image = pygame.image.load('images/ship.bmp')
        self.rect = self.image.get_rect()

        # Start each new ship at the bottom center of the screen.
        ❹ self.rect.midbottom = self.screen_rect.midbottom

        ❺ def blitme(self):
            """Draw the ship at its current location."""
            self.screen.blit(self.image, self.rect)
```

Pygame es eficiente porque te permite tratar todos los elementos del juego como rectángulos (*rects*), aunque no tengan exactamente la forma de un rectángulo. Tratar un elemento como un rectángulo es eficiente porque los rectángulos son formas geométricas simples. Cuando Pygame necesita averiguar si dos elementos del juego han colisionado, por ejemplo, puede hacerlo más rápidamente si trata cada objeto como un rectángulo. Este enfoque suele funcionar lo suficientemente bien como para que nadie que juegue a se dé cuenta de que no estamos trabajando con la forma exacta de cada elemento del juego. En esta clase trataremos la nave y la pantalla como rectángulos.

Pygame is efficient because it lets you treat all game elements like rectangles (*rects*), even if they're not exactly shaped like rectangles. Treating an element as a rectangle is efficient because rectangles are simple geometric shapes. When Pygame needs to figure out whether two game elements have collided, for example, it can do this more quickly if it treats each object as a rectangle. This approach usually works well enough that no one playing the game will notice that we're not working with the exact shape of each game element. We'll treat the ship and the screen as rectangles in this class.

Importamos el módulo `pygame` antes de definir la clase. El método `__init__()` de `Ship` toma dos parámetros: la referencia `self` y una referencia a la instancia actual de la clase `AlienInvasion`. Esto dará a `Ship` acceso a todos los recursos del juego definidos en `AlienInvasion`. A continuación, asignamos la pantalla a un atributo de `Ship` ❶, para que podamos acceder a ella fácilmente en todos los métodos de esta clase. Accedemos al atributo `rect` de la pantalla mediante el método `get_rect()` y lo asignamos a `self.screen_rect` ❷. Esto nos permite colocar la nave en el lugar correcto de la pantalla.

We import the `pygame` module before defining the class. The `__init__()` method of `Ship` takes two parameters: the `self` reference and a reference to the current instance of the `AlienInvasion` class. This will give `Ship` access to all the game resources defined in `AlienInvasion`. We then assign the screen to an attribute of `Ship` ❶, so we can access it easily in all the methods in this class. We access the screen's `rect` attribute using the `get_rect()` method and assign it to `self.screen_rect` ❷. Doing so allows us to place the ship in the correct location on the screen.

Para cargar la imagen, llamamos a `pygame.image.load()` ❸ y le damos la ubicación de la imagen de nuestro barco. Esta función devuelve una superficie que representa el barco, que asignamos a `self.image`. Cuando la imagen está cargada, llamamos a `get_rect()` para acceder al atributo `rect` de la superficie del barco, de modo que podamos utilizarlo posteriormente para colocar el barco.

To load the image, we call `pygame.image.load()` ❸ and give it the location of our ship image. This function returns a surface representing the ship, which we assign to `self.image`. When the image is loaded, we call `get_rect()` to access the ship surface's `rect` attribute so we can later use it to place the ship.

Cuando trabajes con un objeto `rect`, puedes utilizar las coordenadas `x` - `y` - de los bordes superior, inferior, izquierdo y derecho del rectángulo, así como el centro, para colocar el objeto. Puedes fijar cualquiera de estos valores para establecer la posición actual del `rect`. Cuando estés centrando un elemento del juego, trabaja con los atributos `center`, `centerx` o `centery` de un `rect`. Cuando estés trabajando en un borde de la pantalla, trabaja con los atributos `top`, `bottom`, `left` o `right`. También hay atributos que combinan estas propiedades, como `midbottom`, `midtop`, `midleft`, y `midright`. Cuando estés ajustando la colocación horizontal o vertical de `rect`, puedes utilizar simplemente los atributos `x` y `y`, que son las coordenadas `x` - `y` - de su esquina superior izquierda. Estos atributos te evitan tener que hacer cálculos que antes tenían que hacer manualmente los desarrolladores de juegos, y los utilizarás a menudo.

When you're working with a `rect` object, you can use the `x` - and `y` -coordinates of the top, bottom, left, and right edges of the rectangle, as well as the center, to place the object. You can set any of these values to establish the current position of the `rect` . When you're centering a game element, work with the `center` , `centerx` , or `centery` attributes of a `rect` . When you're working at an edge of the screen, work with the `top` , `bottom` , `left` , or `right` attributes. There are also attributes that combine these properties, such as `midbottom` , `midtop` , `midleft` , and `midright` . When you're adjusting the horizontal or vertical placement of the `rect` , you can just use the `x` and `y` attributes, which are the `x` - and `y` -coordinates of its top-left corner. These attributes spare you from having to do calculations that game developers formerly had to do manually, and you'll use them often.

Nota (Note)

En Pygame, el origen (0, 0) está en la esquina superior izquierda de la pantalla, y las coordenadas aumentan a medida que vas hacia abajo y hacia la derecha. En una pantalla de 1200×800, el origen está en la esquina superior izquierda, y la esquina inferior derecha tiene las coordenadas (1200, 800). Estas coordenadas se refieren a la ventana del juego, no a la pantalla física.

In Pygame, the origin (0, 0) is at the top-left corner of the screen, and coordinates increase as you go down and to the right. On a 1200×800 screen, the origin is at the top-left corner, and the bottom-right corner has the coordinates (1200, 800). These coordinates refer to the game window, not the physical screen.

Posicionaremos la nave en el centro inferior de la pantalla. Para ello, haz que el valor de `self.rect.midbottom` coincida con el atributo `midbottom` de la pantalla `rect` ④ . Pygame utiliza estos atributos `rect` para posicionar la imagen de la nave de forma que quede centrada horizontalmente y alineada con la parte inferior de la pantalla.

We'll position the ship at the bottom center of the screen. To do so, make the value of `self.rect.midbottom` match the `midbottom` attribute of the screen's `rect` ④ . Pygame uses these `rect` attributes to position the ship image so it's centered horizontally and aligned with the bottom of the screen.

Por último, definimos el método `blitme()` ⑤ , que dibuja la imagen en la pantalla en la posición especificada por `self.rect` .

Finally, we define the `blitme()` method ⑤ , which draws the image to the screen at the position specified by `self.rect` .

Dibujar el barco en la pantalla (*Drawing the Ship to the Screen*)

Ahora actualicemos *alien_invasion.py* para que cree una nave y llame al método `blitme()` de la nave:

Now let's update *alien_invasion.py* so it creates a ship and calls the ship's `blitme()` method:

alien_invasion.py

alien_invasion.py

```
--snip--
from settings import Settings
from ship import Ship

class AlienInvasion:
    """Overall class to manage game assets and behavior."""

    def __init__(self):
        --snip--
        pygame.display.set_caption("Alien Invasion")

    ① self.ship = Ship(self)

    def run_game(self):
        --snip--
        # Redraw the screen during each pass through the loop.
        self.screen.fill(self.settings.bg_color)
```

```
❷ self.ship.blitme()

# Make the most recently drawn screen visible.
pygame.display.flip()
self.clock.tick(60)
--snip--
```

Importamos `Ship` y creamos una instancia de `Ship` una vez creada la pantalla ❶. La llamada a `Ship()` requiere un argumento: una instancia de `AlienInvasion`. El argumento `self` se refiere aquí a la instancia actual de `AlienInvasion`. Éste es el parámetro que da acceso a `Ship` a los recursos del juego, como el objeto `screen`. Asignamos esta instancia de `Ship` a `self.ship`.

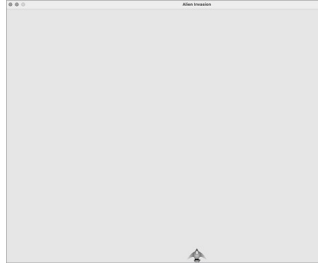
We import `Ship` and then make an instance of `Ship` after the screen has been created ❶. The call to `Ship()` requires one argument: an instance of `AlienInvasion`. The `self` argument here refers to the current instance of `AlienInvasion`. This is the parameter that gives `Ship` access to the game's resources, such as the `screen` object. We assign this `Ship` instance to `self.ship`.

Después de rellenar el fondo, dibujamos la nave en la pantalla llamando a `ship.blitme()`, para que la nave aparezca sobre el fondo ❷.

After filling the background, we draw the ship on the screen by calling `ship.blitme()`, so the ship appears on top of the background ❷.

Cuando ejecutes ahora *alien_invasion.py*, deberías ver una pantalla de juego vacía con el cohete espacial en la parte inferior central, como se muestra en la [Figura 12-2](#).

When you run *alien_invasion.py* now, you should see an empty game screen with the rocket ship sitting at the bottom center, as shown in [Figure 12-2](#).



DESCRIPCIÓN DE LA IMAGEN (En español):

La imagen muestra una pantalla de computadora con un borde gris claro. En la parte superior de la pantalla, hay un título que dice "Invasión Extraterrestre". La mayor parte de la pantalla está vacía y de color gris claro. En la parte inferior, hay una pequeña nave espacial de forma triangular, situada en el centro.

Figura 12-2 : Alien Invasion con la nave en la parte inferior central de la pantalla

Figure 12-2 : Alien Invasion with the ship at the bottom center of the screen

Refactorización: Los métodos `_check_events()` y `_update_screen()` (Refactoring: The `_check_events()` and `_update_screen()` Methods)

En los proyectos grandes, a menudo refactorizarás el código que has escrito antes de añadir más código. La refactorización simplifica la estructura del código que ya has escrito, facilitando su desarrollo. En esta sección, dividiremos el método `run_game()`, que se está haciendo largo, en dos métodos de ayuda. Un *helper method* funciona dentro de una clase, pero no está pensado para ser utilizado por código fuera de la clase. En Python, un guión bajo inicial indica un método de ayuda.

In large projects, you'll often refactor code you've written before adding more code. Refactoring simplifies the structure of the code you've already written, making it easier to build on. In this section, we'll break the `run_game()` method, which is getting lengthy, into two helper methods. A *helper method* does work inside a class but isn't meant to be used by code outside the class. In Python, a single leading underscore indicates a helper method.

El método `_check_events()` (The `_check_events()` Method)

Trasladaremos el código que gestiona los eventos a un método independiente llamado `_check_events()`. Esto simplificará `run_game()` y aislará el bucle de gestión de eventos. Aislar el bucle de eventos te permite gestionar los eventos por separado de otros aspectos del juego, como la actualización de la pantalla.

We'll move the code that manages events to a separate method called `_check_events()` . This will simplify `run_game()` and isolate the event management loop. Isolating the event loop allows you to manage events separately from other aspects of the game, such as updating the screen.

Aquí tienes la clase `AlienInvasion` con el nuevo método `_check_events()` , que sólo afecta al código de `run_game()` :

Here's the `AlienInvasion` class with the new `_check_events()` method, which only affects the code in `run_game()` :

alien_invasion.py

alien_invasion.py

```
def run_game(self):
    """Start the main loop for the game."""
    while True:
        ❶ self._check_events()

        # Redraw the screen during each pass through the loop.
        --snip--

        ❷ def _check_events(self):
            """Respond to keypresses and mouse events."""
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    sys.exit()
```

Creamos un nuevo método `_check_events()` ❷ y trasladamos las líneas que comprueban si el jugador ha hecho clic para cerrar la ventana a este nuevo método.

We make a new `_check_events()` method ❷ and move the lines that check whether the player has clicked to close the window into this new method.

Para llamar a un método desde dentro de una clase, utiliza la notación con puntos con la variable `self` y el nombre del método ❶ . Llamamos al método desde dentro del bucle `while` en `run_game()` .

To call a method from within a class, use dot notation with the variable `self` and the name of the method ❶ . We call the method from inside the `while` loop in `run_game()` .

El método `_actualizar_pantalla()` (*The `_update_screen()` Method*)

Para simplificar aún más `run_game()` , trasladaremos el código para actualizar la pantalla a un método independiente llamado `_update_screen()` :

To further simplify `run_game()` , we'll move the code for updating the screen to a separate method called `_update_screen()` :

`invasión_extraterrestre.py`

`alien_invasion.py`

```
def run_game(self):
    """Start the main loop for the game."""
    while True:
        self._check_events()
        self._update_screen()
        self.clock.tick(60)

def _check_events(self):
    --snip--

def _update_screen(self):
    """Update images on the screen, and flip to the new screen."""
    self.screen.fill(self.settings.bg_color)
    self.ship.blitme()

    pygame.display.flip()
```


Trasladamos el código que dibuja el fondo y la nave y voltea la pantalla a `_update_screen()`. Ahora el cuerpo del bucle principal en `run_game()` es mucho más sencillo. Es fácil ver que buscamos nuevos eventos, actualizamos la pantalla y hacemos tick en el reloj en cada pasada por el bucle.

We moved the code that draws the background and the ship and flips the screen to `_update_screen()`. Now the body of the main loop in `run_game()` is much simpler. It's easy to see that we're looking for new events, updating the screen, and ticking the clock on each pass through the loop.

Si ya has construido varios juegos, probablemente empezarás dividiendo tu código en métodos como éstos. Pero si nunca has abordado un proyecto como éste, es probable que al principio no sepas exactamente cómo estructurar tu código. Este enfoque te da una idea de un proceso de desarrollo realista: empiezas escribiendo tu código de la forma más sencilla posible, y luego lo refactorizas a medida que tu proyecto se hace más complejo.

If you've already built a number of games, you'll probably start out by breaking your code into methods like these. But if you've never tackled a project like this, you probably won't know exactly how to structure your code at first. This approach gives you an idea of a realistic development process: you start out writing your code as simply as possible, and then refactor it as your project becomes more complex.

Ahora que hemos reestructurado el código para que sea más fácil añadir cosas, ¿podemos trabajar en los aspectos dinámicos del juego!

Now that we've restructured the code to make it easier to add to, we can work on the dynamic aspects of the game!

Pruébalo tú mismo (*Try It Yourself*)

12-1. Cielo azul: Crea una ventana Pygame con un fondo azul.

12-1. Blue Sky: Make a Pygame window with a blue background.

12-2. Personaje del juego: Encuentra una imagen de mapa de bits de un personaje de juego que te guste o convierte una imagen a mapa de bits. Crea una clase que dibuje el personaje en el centro de la pantalla, luego haz coincidir el color de fondo de la imagen con el color de fondo de la pantalla o viceversa.

12-2. Game Character: Find a bitmap image of a game character you like or convert an image to a bitmap. Make a class that draws the character at the center of the screen, then match the background color of the image to the background color of the screen or vice versa.

Pilotar la nave (*Piloting the Ship*)

A continuación, daremos al jugador la posibilidad de mover la nave a derecha e izquierda. Escribiremos código que responda cuando el jugador pulse la tecla de flecha derecha o izquierda. Nos centraremos primero en el movimiento hacia la derecha, y luego aplicaremos los mismos principios para controlar el movimiento hacia la izquierda. A medida que vayamos añadiendo este código, aprenderás a controlar el movimiento de las imágenes en la pantalla y a responder a las entradas del usuario.

Next, we'll give the player the ability to move the ship right and left. We'll write code that responds when the player presses the right or left arrow key. We'll focus first on movement to the right, and then we'll apply the same principles to control movement to the left. As we add this code, you'll learn how to control the movement of images on the screen and respond to user input.

Responder a la pulsación de una tecla (*Responding to a Keypress*)

Cada vez que el jugador pulsa una tecla, esa pulsación se registra en Pygame como un evento. Cada evento es recogido por el método `pygame.event.get()`. Tenemos que especificar en nuestro método `_check_events()` qué tipo de eventos queremos que compruebe el juego. Cada pulsación de tecla se registra como un evento `KEYDOWN`.

Whenever the player presses a key, that keypress is registered in Pygame as an event. Each event is picked up by the `pygame.event.get()` method. We need to specify in our

`_check_events()` method what kinds of events we want the game to check for. Each keypress is registered as a `KEYDOWN` event.

Cuando Pygame detecta un evento `KEYDOWN` , tenemos que comprobar si la tecla que se ha pulsado es una que desencadena una determinada acción. Por ejemplo, si el jugador pulsa la tecla de flecha derecha, queremos aumentar el valor `rect.x` de la nave para moverla hacia la derecha:

When Pygame detects a `KEYDOWN` event, we need to check whether the key that was pressed is one that triggers a certain action. For example, if the player presses the right arrow key, we want to increase the ship's `rect.x` value to move the ship to the right:

alien_invasion.py

alien_invasion.py

```
def _check_events(self):
    """Respond to keypresses and mouse events."""
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            sys.exit()
        ❶ elif event.type == pygame.KEYDOWN:
            ❷ if event.key == pygame.K_RIGHT:
                # Move the ship to the right.
                ❸ self.ship.rect.x += 1
```

Dentro de `_check_events()` añadimos un bloque `elif` al bucle de eventos, para responder cuando Pygame detecte un evento `KEYDOWN` ❶ . Comprobamos si la tecla pulsada, `event.key` , es la tecla de la flecha derecha ❷ . La tecla de la flecha derecha está representada por `pygame.K_RIGHT` . Si se ha pulsado la tecla de la flecha derecha, movemos la nave hacia la derecha aumentando el valor de `self.ship.rect.x` en 1 ❸ .

Inside `_check_events()` we add an `elif` block to the event loop, to respond when Pygame detects a `KEYDOWN` event ❶ . We check whether the key pressed, `event.key` , is the right arrow key ❷ . The right arrow key is represented by `pygame.K_RIGHT` . If the right

arrow key was pressed, we move the ship to the right by increasing the value of `self.ship.rect.x` by 1 ③ .

Cuando ejecutes ahora *alien_invasion.py* , la nave debería moverse hacia la derecha un píxel cada vez que pulses la tecla de flecha derecha. Es un comienzo, pero no es una forma eficaz de controlar la nave. Mejoremos este control permitiendo un movimiento continuo.

When you run *alien_invasion.py* now, the ship should move to the right one pixel every time you press the right arrow key. That's a start, but it's not an efficient way to control the ship. Let's improve this control by allowing continuous movement.

Permitir el movimiento continuo (*Allowing Continuous Movement*)

Cuando el jugador mantenga pulsada la tecla de flecha derecha, queremos que la nave siga moviéndose hasta que el jugador suelte la tecla. Haremos que el juego detecte un evento `pygame.KEYUP` para saber cuándo se suelta la tecla de la flecha derecha; entonces utilizaremos los eventos `KEYDOWN` y `KEYUP` junto con una bandera llamada `moving_right` para implementar el movimiento continuo.

When the player holds down the right arrow key, we want the ship to continue moving right until the player releases the key. We'll have the game detect a `pygame.KEYUP` event so we'll know when the right arrow key is released; then we'll use the `KEYDOWN` and `KEYUP` events together with a flag called `moving_right` to implement continuous motion.

Cuando la bandera `moving_right` sea `False` , la nave estará inmóvil. Cuando el jugador pulse la tecla de flecha derecha, pondremos la bandera a `True` , y cuando el jugador suelte la tecla, volveremos a poner la bandera a `False` .

When the `moving_right` flag is `False` , the ship will be motionless. When the player presses the right arrow key, we'll set the flag to `True` , and when the player releases the key, we'll set the flag to `False` again.

La clase `Ship` controla todos los atributos de la nave, así que le daremos un atributo llamado `moving_right` y un método `update()` para comprobar el estado de la bandera

`moving_right` . El método `update()` cambiará la posición de la nave si la bandera se establece en `True` . Llamaremos a este método una vez en cada pasada por el bucle `while` para actualizar la posición de la nave.

The `Ship` class controls all attributes of the ship, so we'll give it an attribute called `moving_right` and an `update()` method to check the status of the `moving_right` flag. The `update()` method will change the position of the ship if the flag is set to `True` . We'll call this method once on each pass through the `while` loop to update the position of the ship.

Estos son los cambios en `Ship` :

Here are the changes to `Ship` :

barco.py

ship.py

```
class Ship:
    """A class to manage the ship."""

    def __init__(self, ai_game):
        --snip--
        # Start each new ship at the bottom center of the screen.
        self.rect.midbottom = self.screen_rect.midbottom

        # Movement flag; start with a ship that's not moving.
        ❶ self.moving_right = False

        ❷ def update(self):
            """Update the ship's position based on the movement flag."""
            if self.moving_right:
                self.rect.x += 1

    def blitme(self):
        --snip--
```

Añadimos un atributo `self.moving_right` en el método `__init__()` y lo establecemos en `False` inicialmente ❶ . Después añadimos `update()` , que mueve el barco hacia la derecha si la bandera es `True` ❷ . El método `update()` será llamado desde fuera de la clase, por lo que no se considera un método de ayuda.

We add a `self.moving_right` attribute in the `__init__()` method and set it to `False` initially ❶ . Then we add `update()` , which moves the ship right if the flag is `True` ❷ . The `update()` method will be called from outside the class, so it's not considered a helper method.

Ahora tenemos que modificar `_check_events()` para que `moving_right` se convierta en `True` cuando se pulse la tecla de flecha derecha y en `False` cuando se suelte la tecla:

Now we need to modify `_check_events()` so that `moving_right` is set to `True` when the right arrow key is pressed and `False` when the key is released:

alien_invasion.py

alien_invasion.py

```
def _check_events(self):
    """Respond to keypresses and mouse events."""
    for event in pygame.event.get():
        --snip--
        elif event.type == pygame.KEYDOWN:
            if event.key == pygame.K_RIGHT:
❶         self.ship.moving_right = True
❷         elif event.type == pygame.KEYUP:
            if event.key == pygame.K_RIGHT:
                self.ship.moving_right = False
```

Aquí modificamos cómo responde el juego cuando el jugador pulsa la tecla de flecha derecha: en lugar de cambiar directamente la posición de la nave, nos limitamos a establecer `moving_right` en `True` ❶ . Luego añadimos un nuevo bloque `elif` , que responde a los eventos `KEYUP` ❷ . Cuando el jugador suelta la tecla de flecha derecha (

K_RIGHT), establecemos moving_right en False .

Here, we modify how the game responds when the player presses the right arrow key: instead of changing the ship's position directly, we merely set moving_right to True ❶ . Then we add a new elif block, which responds to KEYUP events ❷ . When the player releases the right arrow key (K_RIGHT), we set moving_right to False .

A continuación, modificamos el bucle while en run_game() para que llame al método update() de la nave en cada pasada por el bucle:

Next, we modify the while loop in run_game() so it calls the ship's update() method on each pass through the loop:

alien_invasion.py

alien_invasion.py

```
def run_game(self):
    """Start the main loop for the game."""
    while True:
        self._check_events()
        self.ship.update()
        self._update_screen()
        self.clock.tick(60)
```

La posición de la nave se actualizará después de comprobar los eventos del teclado y antes de actualizar la pantalla. Esto permite que la posición de la nave se actualice en respuesta a la entrada del jugador y garantiza que la posición actualizada se utilizará al dibujar la nave en la pantalla.

The ship's position will be updated after we've checked for keyboard events and before we update the screen. This allows the ship's position to be updated in response to player input and ensures the updated position will be used when drawing the ship to the screen.

Cuando ejecutes *alien_invasion.py* y mantengas pulsada la tecla de flecha derecha, la nave debería moverse continuamente hacia la derecha hasta que sueltes la tecla.

When you run *alien_invasion.py* and hold down the right arrow key, the ship should move continuously to the right until you release the key.

Moverse tanto a la izquierda como a la derecha (*Moving Both Left and Right*)

Ahora que la nave puede moverse continuamente hacia la derecha, añadir el movimiento hacia la izquierda es sencillo. De nuevo, modificaremos la clase `Ship` y el método `_check_events()`. Aquí tienes los cambios pertinentes en `__init__()` y `update()` en `Ship`:

Now that the ship can move continuously to the right, adding movement to the left is straightforward. Again, we'll modify the `Ship` class and the `_check_events()` method. Here are the relevant changes to `__init__()` and `update()` in `Ship`:

nave.py

ship.py

```
def __init__(self, ai_game):
    --snip--
    # Movement flags; start with a ship that's not moving.
    self.moving_right = False
    self.moving_left = False

    def update(self):
        """Update the ship's position based on movement flags."""
        if self.moving_right:
            self.rect.x += 1
        if self.moving_left:
            self.rect.x -= 1
```

En `__init__()`, añadimos una bandera `self.moving_left`. En `update()`, utilizamos dos bloques `if` separados, en lugar de un `elif`, para permitir que el valor `rect.x` de la nave aumente y luego disminuya cuando se mantienen pulsadas ambas teclas de flecha. El

resultado es que la nave se queda quieta. Si utilizáramos `elif` para el movimiento hacia la izquierda, la tecla de flecha derecha siempre tendría prioridad. Utilizar dos bloques `if` hace que los movimientos sean más precisos cuando el jugador puede mantener pulsadas momentáneamente ambas teclas al cambiar de dirección.

In `__init__()` , we add a `self.moving_left` flag. In `update()` , we use two separate `if` blocks, rather than an `elif` , to allow the ship's `rect.x` value to be increased and then decreased when both arrow keys are held down. This results in the ship standing still. If we used `elif` for motion to the left, the right arrow key would always have priority. Using two `if` blocks makes the movements more accurate when the player might momentarily hold down both keys when changing directions.

Tenemos que hacer dos añadidos a `_check_events()` :

We have to make two additions to `_check_events()` :

alien_invasion.py

alien_invasion.py

```
def _check_events(self):
    """Respond to keypresses and mouse events."""
    for event in pygame.event.get():
        --snip--
        elif event.type == pygame.KEYDOWN:
            if event.key == pygame.K_RIGHT:
                self.ship.moving_right = True
            elif event.key == pygame.K_LEFT:
                self.ship.moving_left = True

        elif event.type == pygame.KEYUP:
            if event.key == pygame.K_RIGHT:
                self.ship.moving_right = False
            elif event.key == pygame.K_LEFT:
                self.ship.moving_left = False
```

Si se produce un evento `KEYDOWN` para la tecla `K_LEFT` , establecemos `moving_left` en `True` . Si se produce un evento `KEYUP` para la tecla `K_LEFT` , establecemos `moving_left` en `False` . Aquí podemos utilizar bloques `elif` porque cada evento está conectado a una sola tecla. Si el jugador pulsa ambas teclas a la vez, se detectarán dos eventos distintos.

If a `KEYDOWN` event occurs for the `K_LEFT` key, we set `moving_left` to `True` . If a `KEYUP` event occurs for the `K_LEFT` key, we set `moving_left` to `False` . We can use `elif` blocks here because each event is connected to only one key. If the player presses both keys at once, two separate events will be detected.

Cuando ejecutes ahora *alien_invasion.py* , deberías poder mover la nave continuamente a derecha e izquierda. Si mantienes pulsadas ambas teclas, la nave debería dejar de moverse.

When you run *alien_invasion.py* now, you should be able to move the ship continuously to the right and left. If you hold down both keys, the ship should stop moving.

A continuación, perfeccionaremos aún más el movimiento de la nave. Vamos a ajustar la velocidad de la nave y a limitar la distancia a la que puede moverse para que no desaparezca por los lados de la pantalla.

Next, we'll further refine the ship's movement. Let's adjust the ship's speed and limit how far the ship can move so it can't disappear off the sides of the screen.

Ajustar la velocidad de la nave (*Adjusting the Ship's Speed*)

Actualmente, la nave se mueve un píxel por ciclo a través del bucle `while` , pero podemos tener un control más preciso de la velocidad de la nave añadiendo un atributo `ship_speed` a la clase `Settings` . Utilizaremos este atributo para determinar la distancia que debe recorrer la nave en cada pasada por el bucle. Aquí tienes el nuevo atributo en *settings.py* :

Currently, the ship moves one pixel per cycle through the `while` loop, but we can take finer control of the ship's speed by adding a `ship_speed` attribute to the `Settings` class. We'll use this attribute to determine how far to move the ship on each pass through the loop. Here's the new attribute in *settings.py* :

ajustes.py

settings.py

```
class Settings:
    """A class to store all settings for Alien Invasion."""

    def __init__(self):
        --snip--

        # Ship settings
        self.ship_speed = 1.5
```

Fijamos el valor inicial de `ship_speed` en `1.5` . Cuando la nave se mueve ahora, su posición se ajusta en 1,5 píxeles (en lugar de 1 píxel) en cada pasada por el bucle.

We set the initial value of `ship_speed` to `1.5` . When the ship moves now, its position is adjusted by 1.5 pixels (rather than 1 pixel) on each pass through the loop.

Estamos utilizando un flotador para el ajuste de velocidad para tener un control más preciso de la velocidad de la nave cuando aumentemos el ritmo del juego más adelante. Sin embargo, los atributos de `rect` como `x` sólo almacenan valores enteros, así que tenemos que hacer algunas modificaciones en `Ship` :

We're using a float for the speed setting to give us finer control of the ship's speed when we increase the tempo of the game later on. However, `rect` attributes such as `x` store only integer values, so we need to make some modifications to `Ship` :

nave.py

ship.py

```

class Ship:
    """A class to manage the ship."""

    def __init__(self, ai_game):
        """Initialize the ship and set its starting position."""
        self.screen = ai_game.screen
        ❶ self.settings = ai_game.settings
        --snip--

        # Start each new ship at the bottom center of the screen.
        self.rect.midbottom = self.screen_rect.midbottom

        # Store a float for the ship's exact horizontal position.
        ❷ self.x = float(self.rect.x)

        # Movement flags; start with a ship that's not moving.
        self.moving_right = False
        self.moving_left = False

    def update(self):
        """Update the ship's position based on movement flags."""
        # Update the ship's x value, not the rect.
        if self.moving_right:
            ❸ self.x += self.settings.ship_speed
        if self.moving_left:
            self.x -= self.settings.ship_speed

        # Update rect object from self.x.
        ❹ self.rect.x = self.x

    def blitme(self):
        --snip--

```

Creamos un atributo `settings` para `Ship`, de modo que podamos utilizarlo en `update()`

❶ . Como estamos ajustando la posición de la nave en fracciones de píxel, necesitamos asignar la posición a una variable a la que se le pueda asignar un flotador. Puedes utilizar un flotador para ajustar un atributo de `rect`, pero `rect` sólo conservará la parte entera de ese valor. Para hacer un seguimiento preciso de la posición del barco, definimos un nuevo `self.x` ❷ . Utilizamos la función `float()` para convertir el valor de `self.rect.x` en un flotador y asignamos este valor a `self.x` .

We create a `settings` attribute for `Ship` , so we can use it in `update()` ❶ . Because we're adjusting the position of the ship by fractions of a pixel, we need to assign the position to a variable that can have a float assigned to it. You can use a float to set an attribute of a `rect` , but the `rect` will only keep the integer portion of that value. To keep track of the ship's position accurately, we define a new `self.x` ❷ . We use the `float()` function to convert the value of `self.rect.x` to a float and assign this value to `self.x` .

Ahora, cuando cambiamos la posición del barco en `update()` , el valor de `self.x` se ajusta en la cantidad almacenada en `settings.ship_speed` ❸ . Una vez actualizado `self.x` , utilizamos el nuevo valor para actualizar `self.rect.x` , que controla la posición de la nave ❹ . Sólo la parte entera de `self.x` se asignará a `self.rect.x` , pero eso está bien para mostrar la nave.

Now when we change the ship's position in `update()` , the value of `self.x` is adjusted by the amount stored in `settings.ship_speed` ❸ . After `self.x` has been updated, we use the new value to update `self.rect.x` , which controls the position of the ship ❹ . Only the integer portion of `self.x` will be assigned to `self.rect.x` , but that's fine for displaying the ship.

Ahora podemos cambiar el valor de `ship_speed` , y cualquier valor mayor que 1 hará que la nave se mueva más rápido. Esto ayudará a que la nave responda con suficiente rapidez para derribar alienígenas, y nos permitirá cambiar el tempo del juego a medida que el jugador progresa en la partida.

Now we can change the value of `ship_speed` , and any value greater than 1 will make the ship move faster. This will help make the ship respond quickly enough to shoot down aliens, and it will let us change the tempo of the game as the player progresses in gameplay.

Limitar el alcance de la nave (*Limiting the Ship's Range*)

En este punto, la nave desaparecerá de cualquiera de los bordes de la pantalla si mantienes pulsada una tecla de flecha el tiempo suficiente. Vamos a corregir esto para que la nave deje de moverse cuando llegue al borde de la pantalla. Lo haremos modificando el método `update()` de `Ship` :

At this point, the ship will disappear off either edge of the screen if you hold down an arrow key long enough. Let's correct this so the ship stops moving when it reaches the screen's edge. We do this by modifying the `update()` method in `Ship` :

nave.py

ship.py

```
def update(self):
    """Update the ship's position based on movement flags."""
    # Update the ship's x value, not the rect.
    ❶ if self.moving_right and self.rect.right < self.screen_rect.right:
        self.x += self.settings.ship_speed
    ❷ if self.moving_left and self.rect.left > 0:
        self.x -= self.settings.ship_speed

    # Update rect object from self.x.
    self.rect.x = self.x
```

Este código comprueba la posición de la nave antes de cambiar el valor de `self.x` . El código `self.rect.right` devuelve la x -coordenada del borde derecho de la nave `rect` . Si este valor es menor que el valor devuelto por `self.screen_rect.right` , la nave no ha alcanzado el borde derecho de la pantalla ❶ . Lo mismo ocurre con el borde izquierdo: si el valor de la coordenada izquierda de `rect` es mayor que 0, la nave no ha alcanzado el borde izquierdo de la pantalla ❷ . Así te aseguras de que la nave está dentro de estos límites antes de ajustar el valor de `self.x` .

This code checks the position of the ship before changing the value of `self.x` . The code `self.rect.right` returns the x -coordinate of the right edge of the ship's `rect` . If this value is less than the value returned by `self.screen_rect.right` , the ship hasn't reached the right edge of the screen ❶ . The same goes for the left edge: if the value of the left side of the `rect` is greater than 0, the ship hasn't reached the left edge of the screen ❷ . This ensures the ship is within these bounds before adjusting the value of `self.x` .

Cuando ejecutes ahora `alien_invasion.py` , la nave debería dejar de moverse en cualquiera de los bordes de la pantalla. Esto está muy bien; todo lo que hemos hecho es añadir una prueba condicional en una sentencia `if` , ¡pero parece como si la nave chocara contra una

pared o un campo de fuerza en cualquiera de los bordes de la pantalla!

When you run *alien_invasion.py* now, the ship should stop moving at either edge of the screen. This is pretty cool; all we've done is add a conditional test in an `if` statement, but it feels like the ship hits a wall or force field at either edge of the screen!

Refactorización de `_check_events()` (*Refactoring `_check_events()`*)

El método `_check_events()` aumentará su longitud a medida que sigamos desarrollando el juego, así que vamos a dividir `_check_events()` en dos métodos distintos: uno que se ocupe de los eventos de `KEYDOWN` y otro que se ocupe de los eventos de `KEYUP` :

The `_check_events()` method will increase in length as we continue to develop the game, so let's break `_check_events()` into two separate methods: one that handles `KEYDOWN` events and another that handles `KEYUP` events:

alien_invasion.py

alien_invasion.py

```
def _check_events(self):
    """Respond to keypresses and mouse events."""
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            sys.exit()
        elif event.type == pygame.KEYDOWN:
            self._check_keydown_events(event)
        elif event.type == pygame.KEYUP:
            self._check_keyup_events(event)

def _check_keydown_events(self, event):
    """Respond to keypresses."""
    if event.key == pygame.K_RIGHT:
        self.ship.moving_right = True
    elif event.key == pygame.K_LEFT:
```

```

        self.ship.moving_left = True

    def _check_keyup_events(self, event):
        """Respond to key releases."""
        if event.key == pygame.K_RIGHT:
            self.ship.moving_right = False
        elif event.key == pygame.K_LEFT:
            self.ship.moving_left = False

```

Creamos dos nuevos métodos de ayuda: `_check_keydown_events()` y `_check_keyup_events()`. Cada uno necesita un parámetro `self` y un parámetro `event`. Los cuerpos de estos dos métodos están copiados de `_check_events()`, y hemos sustituido el código antiguo por llamadas a los nuevos métodos. El método `_check_events()` es más sencillo ahora con esta estructura de código más limpia, lo que facilitará el desarrollo de otras respuestas a la entrada del jugador.

We make two new helper methods: `_check_keydown_events()` and `_check_keyup_events()`. Each needs a `self` parameter and an `event` parameter. The bodies of these two methods are copied from `_check_events()`, and we've replaced the old code with calls to the new methods. The `_check_events()` method is simpler now with this cleaner code structure, which will make it easier to develop further responses to player input.

Pulsar Q para salir (*Pressing Q to Quit*)

Ahora que respondemos eficazmente a las pulsaciones de teclas, podemos añadir otra forma de salir del juego. Resulta tedioso hacer clic en la X de la parte superior de la ventana del juego para terminar la partida cada vez que pruebas una nueva función, así que añadiremos un atajo de teclado para terminar la partida cuando el jugador pulse Q:

Now that we're responding to keypresses efficiently, we can add another way to quit the game. It gets tedious to click the X at the top of the game window to end the game every time you test a new feature, so we'll add a keyboard shortcut to end the game when the player presses Q:

alien_invasion.py

alien_invasion.py

```
def _check_keydown_events(self, event):
    --snip--
    elif event.key == pygame.K_LEFT:
        self.ship.moving_left = True
    elif event.key == pygame.K_q:
        sys.exit()
```

En `_check_keydown_events()` , añadimos un nuevo bloque que finaliza el juego cuando el jugador pulsa Q. Ahora, cuando estés probando, puedes pulsar Q para cerrar el juego en lugar de utilizar el cursor para cerrar la ventana.

In `_check_keydown_events()` , we add a new block that ends the game when the player presses Q. Now, when testing, you can press Q to close the game instead of using your cursor to close the window.

Ejecutar el juego en modo pantalla completa (*Running the Game in Fullscreen Mode*)

Pygame tiene un modo de pantalla completa que puede gustarte más que ejecutar el juego en una ventana normal. Algunos juegos se ven mejor en el modo de pantalla completa, y en algunos sistemas, el juego puede funcionar mejor en general en el modo de pantalla completa.

Pygame has a fullscreen mode that you might like better than running the game in a regular window. Some games look better in fullscreen mode, and on some systems, the game may perform better overall in fullscreen mode.

Para ejecutar el juego en modo de pantalla completa, realiza los siguientes cambios en `__init__()` :

To run the game in fullscreen mode, make the following changes in `__init__()` :

alien_invasion.py

alien_invasion.py

```
def __init__(self):
    """Initialize the game, and create game resources."""
    pygame.init()
    self.settings = Settings()

    ❶ self.screen = pygame.display.set_mode((0, 0), pygame.FULLSCREEN)
    ❷ self.settings.screen_width = self.screen.get_rect().width
    self.settings.screen_height = self.screen.get_rect().height
    pygame.display.set_caption("Alien Invasion")
```

Al crear la superficie de la pantalla, pasamos un tamaño de (0, 0) y el parámetro `pygame.FULLSCREEN` ❶. Esto le dice a Pygame que calcule un tamaño de ventana que llene la pantalla. Como no conocemos de antemano la anchura y la altura de la pantalla, actualizamos estos parámetros después de crear la pantalla ❷. Utilizamos los atributos `width` y `height` de la pantalla `rect` para actualizar el objeto `settings`.

When creating the screen surface, we pass a size of (0, 0) and the parameter `pygame.FULLSCREEN` ❶. This tells Pygame to figure out a window size that will fill the screen. Because we don't know the width and height of the screen ahead of time, we update these settings after the screen is created ❷. We use the `width` and `height` attributes of the screen's `rect` to update the `settings` object.

Si te gusta cómo se ve o se comporta el juego en el modo de pantalla completa, mantén estos ajustes. Si te gusta más el juego en su propia ventana, puedes volver al planteamiento original en el que fijamos un tamaño de pantalla específico para el juego.

If you like how the game looks or behaves in fullscreen mode, keep these settings. If you liked the game better in its own window, you can revert back to the original approach where we set a specific screen size for the game.

Nota (Note)

Asegúrate de que puedes salir pulsando Q antes de ejecutar el juego en modo de pantalla completa; Pygame no ofrece ninguna forma predeterminada de salir de un juego en modo

de pantalla completa.

Make sure you can quit by pressing Q before running the game in fullscreen mode; Pygame offers no default way to quit a game while in fullscreen mode.

Resumen rápido (A Quick Recap)

En la siguiente sección, añadiremos la posibilidad de disparar balas, lo que implica añadir un nuevo archivo llamado *bullet.py* y hacer algunas modificaciones en algunos de los archivos que ya estamos utilizando. Ahora mismo, tenemos tres archivos que contienen una serie de clases y métodos. Para tener claro cómo está organizado el proyecto, vamos a revisar cada uno de estos archivos antes de añadir más funcionalidad.

In the next section, we'll add the ability to shoot bullets, which involves adding a new file called *bullet.py* and making some modifications to some of the files we're already using. Right now, we have three files containing a number of classes and methods. To be clear about how the project is organized, let's review each of these files before adding more functionality.

alien_invasion.py (*alien_invasion.py*)

El archivo principal, *alien_invasion.py*, contiene la clase `AlienInvasion`. Esta clase crea una serie de atributos importantes que se utilizan a lo largo del juego: los ajustes se asignan a `settings`, la superficie de visualización principal se asigna a `screen`, y también se crea una instancia de `ship` en este archivo. El bucle principal del juego, un bucle `while`, también se almacena en este módulo. El bucle `while` llama a `_check_events()`, `ship.update()`, y `_update_screen()`. También marca el reloj en cada pasada por el bucle.

The main file, *alien_invasion.py*, contains the `AlienInvasion` class. This class creates a number of important attributes used throughout the game: the settings are assigned to `settings`, the main display surface is assigned to `screen`, and a `ship` instance is created in this file as well. The main loop of the game, a `while` loop, is also stored in this module. The `while` loop calls `_check_events()`, `ship.update()`, and `_update_screen()`. It also ticks the clock on each pass through the loop.

El método `_check_events()` detecta eventos relevantes, como pulsaciones de teclas y desbloques, y procesa cada uno de estos tipos de eventos a través de los métodos `_check_keydown_events()` y `_check_keyup_events()` . Por ahora, estos métodos gestionan el movimiento de la nave. La clase `AlienInvasion` también contiene `_update_screen()` , que redibuja la pantalla en cada pasada por el bucle principal.

The `_check_events()` method detects relevant events, such as keypresses and releases, and processes each of these types of events through the methods `_check_keydown_events()` and `_check_keyup_events()` . For now, these methods manage the ship's movement. The `AlienInvasion` class also contains `_update_screen()` , which redraws the screen on each pass through the main loop.

El archivo *alien_invasion.py* es el único que necesitas ejecutar cuando quieras jugar a *Alien Invasion* . Los otros archivos, *settings.py* y *ship.py* , contienen código que se importa en este archivo.

The *alien_invasion.py* file is the only file you need to run when you want to play *Alien Invasion* . The other files, *settings.py* and *ship.py* , contain code that is imported into this file.

ajustes.py (*settings.py*)

El archivo *settings.py* contiene la clase `Settings` . Esta clase sólo tiene un método `__init__()` , que inicializa los atributos que controlan la apariencia del juego y la velocidad de la nave.

The *settings.py* file contains the `Settings` class. This class only has an `__init__()` method, which initializes attributes controlling the game's appearance and the ship's speed.

nave.py (*ship.py*)

El archivo *ship.py* contiene la clase `Ship` . La clase `Ship` tiene un método `__init__()` , un método `update()` para gestionar la posición de la nave, y un método `blitme()` para dibujar la nave en la pantalla. La imagen de la nave se almacena en *ship.bmp* , que está en la carpeta *images* .

The *ship.py* file contains the `Ship` class. The `Ship` class has an `__init__()` method, an `update()` method to manage the ship's position, and a `blitme()` method to draw the ship to the screen. The image of the ship is stored in *ship.bmp*, which is in the *images* folder.

Pruébalo tú mismo (Try It Yourself)

12-3. Documentación de Pygame: Ya hemos avanzado lo suficiente en el juego como para que quieras echar un vistazo a parte de la documentación de Pygame. La página de inicio de Pygame está en <https://pygame.org>, y la página de inicio de la documentación está en <https://pygame.org/docs>. De momento, hojear la documentación. No la necesitarás para completar este proyecto, pero te ayudará si después quieres modificar *Alien Invasion* o crear tu propio juego.

12-3. Pygame Documentation: We're far enough into the game now that you might want to look at some of the Pygame documentation. The Pygame home page is at <https://pygame.org>, and the home page for the documentation is at <https://pygame.org/docs>. Just skim the documentation for now. You won't need it to complete this project, but it will help if you want to modify *Alien Invasion* or make your own game afterward.

12-4. Cohete: Crea un juego que comience con un cohete en el centro de la pantalla. Permite al jugador mover el cohete hacia arriba, abajo, izquierda o derecha utilizando las cuatro teclas de flecha. Asegúrate de que el cohete nunca se mueve más allá de ningún borde de la pantalla.

12-4. Rocket: Make a game that begins with a rocket in the center of the screen. Allow the player to move the rocket up, down, left, or right using the four arrow keys. Make sure the rocket never moves beyond any edge of the screen.

12-5. Teclas: Crea un archivo Pygame que cree una pantalla vacía. En el bucle de eventos, imprime el atributo `event.key` cada vez que se detecte un evento `pygame.KEYDOWN`. Ejecuta el programa y pulsa varias teclas para ver cómo responde Pygame.

12-5. Keys: Make a Pygame file that creates an empty screen. In the event loop, print the `event.key` attribute whenever a `pygame.KEYDOWN` event is detected. Run the program and press various keys to see how Pygame responds.

Disparando balas (*Shooting Bullets*)

Ahora vamos a añadir la posibilidad de disparar balas. Escribiremos un código que dispare una bala, representada por un pequeño rectángulo, cuando el jugador pulse la barra espaciadora. Las balas se desplazarán en línea recta por la pantalla hasta que desaparezcan de la parte superior de la pantalla.

Now let's add the ability to shoot bullets. We'll write code that fires a bullet, which is represented by a small rectangle, when the player presses the spacebar. Bullets will then travel straight up the screen until they disappear off the top of the screen.

Añadir la configuración de las balas (*Adding the Bullet Settings*)

Al final del método `__init__()`, actualizaremos *settings.py* para incluir los valores que necesitaremos para una nueva clase `Bullet`:

At the end of the `__init__()` method, we'll update *settings.py* to include the values we'll need for a new `Bullet` class:

settings.py

settings.py

```
def __init__(self):
    --snip--
    # Bullet settings
    self.bullet_speed = 2.0
    self.bullet_width = 3
```


② `self.rect.midtop = ai_game.ship.rect.midtop`

`# Store the bullet's position as a float.`

③ `self.y = float(self.rect.y)`

La clase `Bullet` hereda de `Sprite`, que importamos del módulo `pygame.sprite`. Cuando utilizas sprites, puedes agrupar elementos relacionados en tu juego y actuar sobre todos los elementos agrupados a la vez. Para crear una instancia de bala, `__init__()` necesita la instancia actual de `AlienInvasion`, y llamamos a `super()` para heredar correctamente de `Sprite`. También establecemos atributos para los objetos pantalla y ajustes, y para el color de la bala.

The `Bullet` class inherits from `Sprite`, which we import from the `pygame.sprite` module. When you use sprites, you can group related elements in your game and act on all the grouped elements at once. To create a bullet instance, `__init__()` needs the current instance of `AlienInvasion`, and we call `super()` to inherit properly from `Sprite`. We also set attributes for the screen and settings objects, and for the bullet's color.

A continuación creamos el atributo `rect` de la viñeta ①. La bala no se basa en una imagen, así que tenemos que construir una `rect` desde cero utilizando la clase `pygame.Rect()`. Esta clase necesita las coordenadas `x` - `y` - de la esquina superior izquierda del `rect`, y la anchura y altura del `rect`. Inicializamos el `rect` en `(0, 0)`, pero lo moveremos a la ubicación correcta en la línea siguiente, porque la posición de la bala depende de la posición del barco. Obtenemos la anchura y la altura de la bala a partir de los valores almacenados en `self.settings`.

Next we create the bullet's `rect` attribute ①. The bullet isn't based on an image, so we have to build a `rect` from scratch using the `pygame.Rect()` class. This class requires the `x` - and `y` - coordinates of the top-left corner of the `rect`, and the width and height of the `rect`. We initialize the `rect` at `(0, 0)`, but we'll move it to the correct location in the next line, because the bullet's position depends on the ship's position. We get the width and height of the bullet from the values stored in `self.settings`.

Ajustamos el atributo `midtop` de la bala para que coincida con el atributo `midtop` de la nave ②. Esto hará que la bala emerja de la parte superior de la nave, haciendo que parezca que la bala se dispara desde la nave. Utilizamos un flotador para la coordenada `y` de la bala, de modo que podamos realizar ajustes finos en la velocidad de la bala ③.

We set the bullet's `midtop` attribute to match the ship's `midtop` attribute ❷ . This will make the bullet emerge from the top of the ship, making it look like the bullet is fired from the ship. We use a float for the bullet's `y` -coordinate so we can make fine adjustments to the bullet's speed ❸ .

Aquí tienes la segunda parte de *bullet.py* , `update()` y `draw_bullet()` :

Here's the second part of *bullet.py* , `update()` and `draw_bullet()` :

bullet.py

bullet.py

```
def update(self):
    """Move the bullet up the screen."""
    # Update the exact position of the bullet.
    ❶ self.y -= self.settings.bullet_speed
    # Update the rect position.
    ❷ self.rect.y = self.y

def draw_bullet(self):
    """Draw the bullet to the screen."""
    ❸ pygame.draw.rect(self.screen, self.color, self.rect)
```

El método `update()` gestiona la posición de la bala. Cuando se dispara una bala, se desplaza hacia arriba en la pantalla, lo que corresponde a un valor de coordenada `y` decreciente. Para actualizar la posición, restamos la cantidad almacenada en `settings.bullet_speed` de `self.y` ❶ . A continuación, utilizamos el valor de `self.y` para establecer el valor de `self.rect.y` ❷ .

The `update()` method manages the bullet's position. When a bullet is fired, it moves up the screen, which corresponds to a decreasing `y` -coordinate value. To update the position, we subtract the amount stored in `settings.bullet_speed` from `self.y` ❶ . We then use the value of `self.y` to set the value of `self.rect.y` ❷ .

El ajuste de `bullet_speed` nos permite aumentar la velocidad de las balas a medida que avanza el juego o según sea necesario para refinar el comportamiento del juego. Una vez disparada una bala, nunca cambiamos el valor de su `x`-coordenada, por lo que viajará verticalmente en línea recta aunque la nave se mueva.

The `bullet_speed` setting allows us to increase the speed of the bullets as the game progresses or as needed to refine the game's behavior. Once a bullet is fired, we never change the value of its `x`-coordinate, so it will travel vertically in a straight line even if the ship moves.

Cuando queremos dibujar una bala, llamamos a `draw_bullet()`. La función `draw.rect()` rellena la parte de la pantalla definida por la bala `rect` con el color almacenado en `self.color` ❸.

When we want to draw a bullet, we call `draw_bullet()`. The `draw.rect()` function fills the part of the screen defined by the bullet's `rect` with the color stored in `self.color` ❸.

Almacenar viñetas en un grupo (*Storing Bullets in a Group*)

Ahora que tenemos una clase `Bullet` y los ajustes necesarios definidos, podemos escribir código para disparar una bala cada vez que el jugador pulse la barra espaciadora. Crearemos un grupo en `AlienInvasion` para almacenar todas las balas activas, de modo que podamos gestionar las balas que ya han sido disparadas. Este grupo será una instancia de la clase `pygame.sprite.Group`, que se comporta como una lista con algunas funciones adicionales que resultan útiles cuando se construyen juegos. Utilizaremos este grupo para dibujar las balas en la pantalla en cada pasada por el bucle principal y para actualizar la posición de cada bala.

Now that we have a `Bullet` class and the necessary settings defined, we can write code to fire a bullet each time the player presses the spacebar. We'll create a group in `AlienInvasion` to store all the active bullets so we can manage the bullets that have already been fired. This group will be an instance of the `pygame.sprite.Group` class, which behaves like a list with some extra functionality that's helpful when building games. We'll use this group to draw bullets to the screen on each pass through the main loop and to update each bullet's position.

En primer lugar, importaremos la nueva clase `Bullet` :

First, we'll import the new `Bullet` class:

alien_invasion.py

alien_invasion.py

```
--snip--  
from ship import Ship  
from bullet import Bullet
```

A continuación crearemos el grupo que contiene las balas en `__init__()` :

Next we'll create the group that holds the bullets in `__init__()` :

invasión_extraterrestre.py

alien_invasion.py

```
def __init__(self):  
    --snip--  
    self.ship = Ship(self)  
    self.bullets = pygame.sprite.Group()
```

Después tendremos que actualizar la posición de las balas en cada pasada por el bucle `while` :

Then we need to update the position of the bullets on each pass through the `while` loop:

alien_invasion.py

alien_invasion.py

```
def run_game(self):
    """Start the main loop for the game."""
    while True:
        self._check_events()
        self.ship.update()
        self.bullets.update()
        self._update_screen()
        self.clock.tick(60)
```

Cuando llamas a `update()` en un grupo, el grupo llama automáticamente a `update()` por cada sprite del grupo. La línea `self.bullets.update()` llama a `bullet.update()` por cada bala que colocamos en el grupo `bullets` .

When you call `update()` on a group, the group automatically calls `update()` for each sprite in the group. The line `self.bullets.update()` calls `bullet.update()` for each bullet we place in the group `bullets` .

Disparar balas (*Firing Bullets*)

En `AlienInvasion` , tenemos que modificar `_check_keydown_events()` para disparar una bala cuando el jugador pulse la barra espaciadora. No necesitamos modificar `_check_keyup_events()` porque no ocurre nada cuando se suelta la barra espaciadora. También tenemos que modificar `_update_screen()` para asegurarnos de que cada bala se dibuja en la pantalla antes de llamar a `flip()` .

In `AlienInvasion` , we need to modify `_check_keydown_events()` to fire a bullet when the player presses the spacebar. We don't need to change `_check_keyup_events()` because nothing happens when the spacebar is released. We also need to modify `_update_screen()` to make sure each bullet is drawn to the screen before we call `flip()` .

Habrá un poco de trabajo que hacer cuando disparemos una bala, así que vamos a escribir un nuevo método, `_fire_bullet()` , para manejar este trabajo:

There will be a bit of work to do when we fire a bullet, so let's write a new method, `_fire_bullet()` , to handle this work:

alien_invasion.py

alien_invasion.py

```
def _check_keydown_events(self, event):
    --snip--
    elif event.key == pygame.K_q:
        sys.exit()
    ❶ elif event.key == pygame.K_SPACE:
        self._fire_bullet()

def _check_keyup_events(self, event):
    --snip--

def _fire_bullet(self):
    """Create a new bullet and add it to the bullets group."""
    ❷ new_bullet = Bullet(self)
    ❸ self.bullets.add(new_bullet)

def _update_screen(self):
    """Update images on the screen, and flip to the new screen."""
    self.screen.fill(self.settings.bg_color)
    ❹ for bullet in self.bullets.sprites():
        bullet.draw_bullet()
    self.ship.blitme()

    pygame.display.flip()
--snip--
```

Llamamos a `_fire_bullet()` cuando se pulsa la barra espaciadora ❶ . En `_fire_bullet()` , creamos una instancia de `Bullet` y la llamamos `new_bullet` ❷ . Luego lo añadimos al grupo `bullets` utilizando el método `add()` ❸ . El método `add()` es similar a `append()` , pero está escrito específicamente para grupos Pygame.

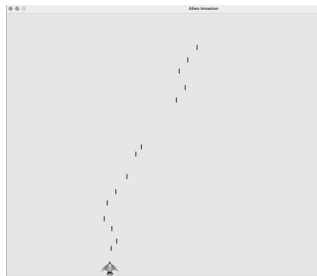
We call `_fire_bullet()` when the spacebar is pressed ❶. In `_fire_bullet()`, we make an instance of `Bullet` and call it `new_bullet` ❷. We then add it to the group `bullets` using the `add()` method ❸. The `add()` method is similar to `append()`, but it's written specifically for Pygame groups.

El método `bullets.sprites()` devuelve una lista de todos los sprites del grupo `bullets`. Para dibujar todas las balas disparadas en la pantalla, hacemos un bucle a través de los sprites en `bullets` y llamamos a `draw_bullet()` en cada uno de ellos ❹. Colocamos este bucle antes de la línea que dibuja la nave, para que las balas no empiecen encima de la nave.

The `bullets.sprites()` method returns a list of all sprites in the group `bullets`. To draw all fired bullets to the screen, we loop through the sprites in `bullets` and call `draw_bullet()` on each one ❹. We place this loop before the line that draws the ship, so the bullets don't start out on top of the ship.

Cuando ejecutes ahora `alien_invasion.py`, deberías poder mover la nave a derecha e izquierda y disparar tantas balas como quieras. Las balas suben por la pantalla y desaparecen cuando llegan arriba, como se muestra en la [Figura 12-3](#). Puedes modificar el tamaño, el color y la velocidad de las balas en `settings.py`.

When you run `alien_invasion.py` now, you should be able to move the ship right and left and fire as many bullets as you want. The bullets travel up the screen and disappear when they reach the top, as shown in [Figure 12-3](#). You can alter the size, color, and speed of the bullets in `settings.py`.



DESCRIPCIÓN DE LA IMAGEN (En español):

La imagen muestra un gráfico en blanco y negro que representa una escena de un videojuego titulado "Invasión Alienígena". En la parte inferior aparece una pequeña nave espacial con la que el jugador parece

estar interactuando,
disparando proyectiles
o láseres hacia el cielo.
Estos disparos están
representados por
líneas verticales que se
dirigen hacia la parte
superior de la imagen.
El fondo es gris y no
hay otros elementos ni
detalles adicionales
visibles en la escena.

Figura 12-3 : La nave tras disparar una serie de balas

Figure 12-3 : The ship after firing a series of bullets

Borrar balas antiguas (Deleting Old Bullets)

De momento, las balas desaparecen cuando llegan arriba, pero sólo porque Pygame no puede dibujarlas por encima de la parte superior de la pantalla. En realidad, las balas siguen existiendo, sólo que sus valores de coordenadas `y` son cada vez más negativos. Esto es un problema porque siguen consumiendo memoria y capacidad de proceso.

At the moment, the bullets disappear when they reach the top, but only because Pygame can't draw them above the top of the screen. The bullets actually continue to exist; their `y`-coordinate values just grow increasingly negative. This is a problem because they continue to consume memory and processing power.

Tenemos que deshacernos de estas viejas balas, o el juego se ralentizará por hacer tanto trabajo innecesario. Para ello, necesitamos detectar cuándo el valor `bottom` de la `rect` de una bala tiene un valor de 0, lo que indica que la bala ha pasado de la parte superior de la pantalla:

We need to get rid of these old bullets, or the game will slow down from doing so much unnecessary work. To do this, we need to detect when the `bottom` value of a bullet's `rect` has a value of 0, which indicates the bullet has passed off the top of the screen:

alien_invasion.py

alien_invasion.py

```
def run_game(self):
    """Start the main loop for the game."""
    while True:
        self._check_events()
        self.ship.update()
        self.bullets.update()

        # Get rid of bullets that have disappeared.
        ❶ for bullet in self.bullets.copy():
            ❷ if bullet.rect.bottom <= 0:
                ❸ self.bullets.remove(bullet)
            ❹ print(len(self.bullets))

        self._update_screen()
        self.clock.tick(60)
```

Cuando utilizas un bucle `for` con una lista (o un grupo en Pygame), Python espera que la lista mantenga la misma longitud mientras se ejecute el bucle. Eso significa que no puedes eliminar elementos de una lista o grupo dentro de un bucle `for`, así que tenemos que hacer un bucle sobre una copia del grupo. Utilizamos el método `copy()` para configurar el bucle `for` ❶, que nos deja libertad para modificar el grupo original `bullets` dentro del bucle. Comprobamos cada viñeta para ver si ha desaparecido de la parte superior de la pantalla ❷. Si es así, la eliminamos de `bullets` ❸. Insertamos una llamada a `print()` para mostrar cuántas balas existen actualmente en el juego y verificamos que se eliminan cuando llegan a la parte superior de la pantalla ❹.

When you use a `for` loop with a list (or a group in Pygame), Python expects that the list will stay the same length as long as the loop is running. That means you can't remove items from a list or group within a `for` loop, so we have to loop over a copy of the group. We use the `copy()` method to set up the `for` loop ❶, which leaves us free to modify the original `bullets` group inside the loop. We check each bullet to see whether it has disappeared off the top of the screen ❷. If it has, we remove it from `bullets` ❸. We insert a `print()` call to show how many bullets currently exist in the game and verify they're being deleted when they reach the top of the screen ❹.

Si este código funciona correctamente, podemos observar la salida del terminal mientras disparamos balas y ver que el número de balas disminuye a cero después de que cada serie de balas haya despejado la parte superior de la pantalla. Después de ejecutar el juego y

comprobar que las balas se eliminan correctamente, elimina la llamada a `print()` . Si la dejas, el juego se ralentizará significativamente porque se tarda más tiempo en escribir la salida en el terminal que en dibujar los gráficos en la ventana del juego.

If this code works correctly, we can watch the terminal output while firing bullets and see that the number of bullets decreases to zero after each series of bullets has cleared the top of the screen. After you run the game and verify that bullets are being deleted properly, remove the `print()` call. If you leave it in, the game will slow down significantly because it takes more time to write output to the terminal than it does to draw graphics to the game window.

Limitar el número de balas (*Limiting the Number of Bullets*)

Muchos juegos de disparos limitan el número de balas que un jugador puede tener en la pantalla a la vez; al hacerlo, animan a los jugadores a disparar con precisión. Haremos lo mismo en *Alien Invasion* .

Many shooting games limit the number of bullets a player can have on the screen at one time; doing so encourages players to shoot accurately. We'll do the same in *Alien Invasion* .

Primero, almacena el número de balas permitidas en *settings.py* :

First, store the number of bullets allowed in *settings.py* :

settings.py

settings.py

```
# Bullet settings
--snip--
self.bullet_color = (60, 60, 60)
self.bullets_allowed = 3
```

Esto limita al jugador a tres balas a la vez. Utilizaremos esta configuración en `AlienInvasion` para comprobar cuántas balas existen antes de crear una nueva bala en `_fire_bullet()` :

This limits the player to three bullets at a time. We'll use this setting in `AlienInvasion` to check how many bullets exist before creating a new bullet in `_fire_bullet()` :

alien_invasion.py

alien_invasion.py

```
def _fire_bullet(self):
    """Create a new bullet and add it to the bullets group."""
    if len(self.bullets) < self.settings.bullets_allowed:
        new_bullet = Bullet(self)
        self.bullets.add(new_bullet)
```

Cuando el jugador pulsa la barra espaciadora, comprobamos la longitud de `bullets` . Si `len(self.bullets)` es inferior a tres, creamos una nueva bala. Pero si ya hay tres balas activas, no ocurre nada cuando se pulsa la barra espaciadora. Cuando ejecutes el juego ahora, sólo deberías poder disparar balas en grupos de tres.

When the player presses the spacebar, we check the length of `bullets` . If `len(self.bullets)` is less than three, we create a new bullet. But if three bullets are already active, nothing happens when the spacebar is pressed. When you run the game now, you should only be able to fire bullets in groups of three.

Creación del método `_update_bullets()` (Creating the `_update_bullets()` Method)

Queremos mantener la clase `AlienInvasion` razonablemente bien organizada, así que ahora que hemos escrito y comprobado el código de gestión de las balas, podemos trasladarlo a un método aparte. Crearemos un nuevo método llamado `_update_bullets()` y lo añadiremos justo antes de `_update_screen()` :

We want to keep the `AlienInvasion` class reasonably well organized, so now that we've written and checked the bullet management code, we can move it to a separate method. We'll create a new method called `_update_bullets()` and add it just before `_update_screen()` :

alien_invasion.py

alien_invasion.py

```
def _update_bullets(self):
    """Update position of bullets and get rid of old bullets."""
    # Update bullet positions.
    self.bullets.update()

    # Get rid of bullets that have disappeared.
    for bullet in self.bullets.copy():
        if bullet.rect.bottom <= 0:
            self.bullets.remove(bullet)
```

El código de `_update_bullets()` está cortado y pegado de `run_game()` ; lo único que hemos hecho aquí es aclarar los comentarios.

The code for `_update_bullets()` is cut and pasted from `run_game()` ; all we've done here is clarify the comments.

El bucle `while` de `run_game()` vuelve a parecer sencillo:

The `while` loop in `run_game()` looks simple again:

alien_invasion.py

alien_invasion.py

```
while True:
    self._check_events()
    self.ship.update()
    self._update_bullets()
    self._update_screen()
    self.clock.tick(60)
```

Ahora nuestro bucle principal sólo contiene un código mínimo, por lo que podemos leer rápidamente los nombres de los métodos y comprender lo que ocurre en el juego. El bucle principal comprueba la entrada del jugador, y luego actualiza la posición de la nave y las balas que se hayan disparado. A continuación, utilizamos las posiciones actualizadas para dibujar una nueva pantalla y marcar el reloj al final de cada pasada por el bucle.

Now our main loop contains only minimal code, so we can quickly read the method names and understand what's happening in the game. The main loop checks for player input, and then updates the position of the ship and any bullets that have been fired. We then use the updated positions to draw a new screen and tick the clock at the end of each pass through the loop.

Ejecuta *alien_invasion.py* una vez más, y asegúrate de que aún puedes disparar balas sin errores.

Run *alien_invasion.py* one more time, and make sure you can still fire bullets without errors.

Pruébalo tú mismo (Try It Yourself)

12-6. Tirador lateral: Escribe un juego que coloque una nave en el lado izquierdo de la pantalla y permita al jugador mover la nave arriba y abajo. Haz que la nave dispare una bala que recorra la pantalla hacia la derecha cuando el jugador pulse la barra espaciadora. Asegúrate de que las balas se borran cuando desaparecen de la pantalla.

12-6. Sideways Shooter: Write a game that places a ship on the left side of the screen and allows the player to move the ship up and down. Make the ship fire a bullet that travels

right across the screen when the player presses the spacebar. Make sure bullets are deleted once they disappear off the screen.

Resumen (Summary)

En este capítulo, aprendiste a hacer un plan para un juego y conociste la estructura básica de un juego escrito en Pygame. Aprendiste a establecer un color de fondo y a almacenar los ajustes en una clase separada, donde puedes ajustarlos más fácilmente. Viste cómo dibujar una imagen en la pantalla y dar al jugador el control sobre el movimiento de los elementos del juego. Creaste elementos que se mueven solos, como balas que vuelan por la pantalla, y eliminaste objetos que ya no son necesarios. También aprendiste a refactorizar el código de un proyecto de forma regular para facilitar el desarrollo continuo.

In this chapter, you learned to make a plan for a game and learned the basic structure of a game written in Pygame. You learned to set a background color and store settings in a separate class where you can adjust them more easily. You saw how to draw an image to the screen and give the player control over the movement of game elements. You created elements that move on their own, like bullets flying up a screen, and you deleted objects that are no longer needed. You also learned to refactor code in a project on a regular basis to facilitate ongoing development.

En el Capítulo 13 , añadiremos extraterrestres a *Alien Invasion* . Al final del capítulo, serás capaz de derribar alienígenas, ¡esperemos que antes de que lleguen a tu nave!

In Chapter 13 , we'll add aliens to *Alien Invasion* . By the end of the chapter, you'll be able to shoot down aliens, hopefully before they reach your ship!

Capítulo 13: ¡Alienígenas! (Chapter 13: Aliens!)

13

¡Extraterrestres! (13 Aliens!)



DESCRIPCIÓN DE LA
IMAGEN (En español):

La imagen muestra una ilustración de una serpiente que está enrollada alrededor de un motor de chorro. El fondo es de un tono gris claro, lo que hace resaltar los detalles del dibujo en blanco y negro. La serpiente tiene una expresión particular, con unos ojos grandes y una sonrisa, mientras que el motor de chorro sobresale por detrás de su cuerpo, mostrando claramente sus detalles mecánicos.

En este capítulo, añadiremos alienígenas a *Alien Invasion* . Añadiremos un alienígena cerca de la parte superior de la pantalla y luego generaremos toda una flota de alienígenas. Haremos que la flota avance hacia los lados y hacia abajo, y nos desharemos de cualquier alienígena alcanzado por una bala. Por último, limitaremos el número de naves del jugador y terminaremos la partida cuando el jugador se quede sin naves.

In this chapter, we'll add aliens to *Alien Invasion* . We'll add one alien near the top of the screen and then generate a whole fleet of aliens. We'll make the fleet advance sideways and down, and we'll get rid of any aliens hit by a bullet. Finally, we'll limit the number of ships a player has and end the game when the player runs out of ships.

A medida que trabajes en este capítulo, aprenderás más sobre Pygame y sobre la gestión de un gran proyecto. También aprenderás a detectar colisiones entre objetos del juego, como balas y alienígenas. Detectar colisiones te ayuda a definir las interacciones entre los elementos de tus juegos. Por ejemplo, puedes confinar a un personaje dentro de las paredes

de un laberinto o pasar una pelota entre dos personajes. Seguiremos trabajando a partir de un plan que revisaremos de vez en cuando para mantener el enfoque de nuestras sesiones de escritura de código.

As you work through this chapter, you'll learn more about Pygame and about managing a large project. You'll also learn to detect collisions between game objects, like bullets and aliens. Detecting collisions helps you define interactions between elements in your games. For example, you can confine a character inside the walls of a maze or pass a ball between two characters. We'll continue to work from a plan that we revisit occasionally to maintain the focus of our code-writing sessions.

Antes de empezar a escribir nuevo código para añadir una flota de alienígenas a la pantalla, echemos un vistazo al proyecto y actualicemos nuestro plan.

Before we start writing new code to add a fleet of aliens to the screen, let's look at the project and update our plan.

Revisar el proyecto *(Reviewing the Project)*

Cuando empiezas una nueva fase de desarrollo en un proyecto grande, siempre es buena idea revisar tu plan y aclarar lo que quieres conseguir con el código que vas a escribir. En este capítulo, haremos lo siguiente:

When you're beginning a new phase of development on a large project, it's always a good idea to revisit your plan and clarify what you want to accomplish with the code you're about to write. In this chapter, we'll do the following:

Listado en español:

- Añade un único alienígena en la esquina superior izquierda de la pantalla, con un espaciado adecuado a su alrededor.
- Llena la parte superior de la pantalla con tantos alienígenas como quepan horizontalmente. A continuación, crearemos filas adicionales de alienígenas hasta que tengamos una flota completa.
- Haz que la flota se mueva hacia los lados y hacia abajo hasta que toda la flota sea derribada, un alienígena golpee la nave o un alienígena llegue al suelo. Si toda la

flota es derribada, crearemos una nueva flota. Si un alienígena alcanza la nave o el suelo, destruiremos la nave y crearemos una nueva flota.

- Limitaremos el número de naves que el jugador puede utilizar, y terminaremos la partida cuando el jugador haya agotado el número de naves asignado.

Listing in English:

- Add a single alien to the top-left corner of the screen, with appropriate spacing around it.
- Fill the upper portion of the screen with as many aliens as we can fit horizontally. We'll then create additional rows of aliens until we have a full fleet.
- Make the fleet move sideways and down until the entire fleet is shot down, an alien hits the ship, or an alien reaches the ground. If the entire fleet is shot down, we'll create a new fleet. If an alien hits the ship or the ground, we'll destroy the ship and create a new fleet.
- Limit the number of ships the player can use, and end the game when the player has used up the allotted number of ships.

Perfeccionaremos este plan a medida que implementemos características, pero esto es lo suficientemente específico como para empezar a escribir código.

We'll refine this plan as we implement features, but this is specific enough to start writing code.

También deberías revisar el código existente cuando empieces a trabajar en una nueva serie de características de un proyecto. Dado que cada nueva fase suele hacer más complejo un proyecto, es mejor limpiar cualquier código desordenado o ineficiente. Hemos ido refactorizando sobre la marcha, así que no hay código que tengamos que refactorizar en este momento.

You should also review your existing code when you begin working on a new series of features in a project. Because each new phase typically makes a project more complex, it's best to clean up any cluttered or inefficient code. We've been refactoring as we go, so there isn't any code that we need to refactor at this point.

Crear el primer alienígena (*Creating the First Alien*)

Colocar un alienígena en la pantalla es como colocar una nave en la pantalla. El comportamiento de cada alienígena está controlado por una clase llamada `Alien`, que estructuraremos como la clase `Ship`. Seguiremos utilizando imágenes de mapa de bits por

simplicidad. Puedes encontrar tu propia imagen para un alienígena o utilizar la que se muestra en la [Figura 13-1](#) , que está disponible en los recursos del libro en https://ehmatthes.github.io/pcc_3e . Esta imagen tiene un fondo gris, que coincide con el color de fondo de la pantalla. Asegúrate de guardar el archivo de imagen que elijas en la carpeta *images* .

Placing one alien on the screen is like placing a ship on the screen. Each alien's behavior is controlled by a class called `Alien` , which we'll structure like the `Ship` class. We'll continue using bitmap images for simplicity. You can find your own image for an alien or use the one shown in [Figure 13-1](#) , which is available in the book's resources at https://ehmatthes.github.io/pcc_3e . This image has a gray background, which matches the screen's background color. Make sure you save the image file you choose in the *images* folder.



DESCRIPCIÓN DE LA
IMAGEN (En español):

La imagen muestra un dibujo de un objeto en forma de platillo volador con patas. El cuerpo del platillo es circular, con una cúpula en la parte superior. Hay una serie de círculos alineados a lo largo del borde del platillo. Las patas son tres soportes que descienden del cuerpo principal, dándole un aspecto futurista. El diseño es monocromático y tiene un estilo simple, pero distintivo.

Figura 13-1 : El alienígena que utilizaremos para construir la flota

Figure 13-1 : The alien we'll use to build the fleet

Crear la clase Alien (Creating the Alien Class)

Ahora escribiremos la clase `Alien` y la guardaremos como *alien.py* :

Now we'll write the `Alien` class and save it as *alien.py* :

alien.py

alien.py

```
import pygame
from pygame.sprite import Sprite

class Alien(Sprite):
    """A class to represent a single alien in the fleet."""

    def __init__(self, ai_game):
        """Initialize the alien and set its starting position."""
        super().__init__()
        self.screen = ai_game.screen

        # Load the alien image and set its rect attribute.
        self.image = pygame.image.load('images/alien.bmp')
        self.rect = self.image.get_rect()

        # Start each new alien near the top left of the screen.
        ❶ self.rect.x = self.rect.width
        self.rect.y = self.rect.height

        # Store the alien's exact horizontal position.
        ❷ self.x = float(self.rect.x)
```

La mayor parte de esta clase es como la clase `Ship` , excepto por la colocación del alienígena en la pantalla. Inicialmente colocamos cada alienígena cerca de la esquina superior izquierda de la pantalla; añadimos un espacio a su izquierda igual a la anchura del alienígena y un espacio por encima igual a su altura ❶ , para que sea fácil de ver. Nos interesa sobre todo la velocidad horizontal de los alienígenas, así que seguiremos con precisión la posición horizontal de cada alienígena ❷ .

Most of this class is like the `Ship` class, except for the alien's placement on the screen. We initially place each alien near the top-left corner of the screen; we add a space to the left of it that's equal to the alien's width and a space above it equal to its height ❶ , so it's easy to

see. We're mainly concerned with the aliens' horizontal speed, so we'll track the horizontal position of each alien precisely ② .

Esta clase `Alien` no necesita un método para dibujarla en la pantalla; en su lugar, utilizaremos un método de grupo de Pygame que dibuja automáticamente todos los elementos de un grupo en la pantalla.

This `Alien` class doesn't need a method for drawing it to the screen; instead, we'll use a Pygame group method that automatically draws all the elements of a group to the screen.

Crear una instancia del Alien (*Creating an Instance of the Alien*)

Queremos crear una instancia de `Alien` para poder ver al primer alienígena en la pantalla. Como forma parte de nuestro trabajo de configuración, añadiremos el código para esta instancia al final del método `__init__()` en `AlienInvasion` . Con el tiempo, crearemos toda una flota de alienígenas, lo que supondrá bastante trabajo, así que crearemos un nuevo método de ayuda llamado `_create_fleet()` .

We want to create an instance of `Alien` so we can see the first alien on the screen. Because it's part of our setup work, we'll add the code for this instance at the end of the `__init__()` method in `AlienInvasion` . Eventually, we'll create an entire fleet of aliens, which will be quite a bit of work, so we'll make a new helper method called `_create_fleet()` .

El orden de los métodos en una clase no importa, siempre que haya cierta coherencia en su colocación. Yo colocaré `_create_fleet()` justo antes del método `_update_screen()` , pero cualquier lugar de `AlienInvasion` servirá. En primer lugar, importaremos la clase `Alien` .

The order of methods in a class doesn't matter, as long as there's some consistency to how they're placed. I'll place `_create_fleet()` just before the `_update_screen()` method, but anywhere in `AlienInvasion` will work. First, we'll import the `Alien` class.

Aquí tienes las declaraciones `import` actualizadas para *alien_invasion.py* :

Here are the updated `import` statements for *alien_invasion.py* :

alien_invasion.py

alien_invasion.py

```
--snip--  
from bullet import Bullet  
from alien import Alien
```

Y aquí está el método `__init__()` actualizado:

And here's the updated `__init__()` method:

alien_invasion.py

alien_invasion.py

```
def __init__(self):  
    --snip--  
    self.ship = Ship(self)  
    self.bullets = pygame.sprite.Group()  
    self.aliens = pygame.sprite.Group()  
  
    self._create_fleet()
```

Creamos un grupo para alojar la flota de alienígenas, y llamamos a `_create_fleet()` , que estamos a punto de escribir.

We create a group to hold the fleet of aliens, and we call `_create_fleet()` , which we're about to write.

Aquí está el nuevo método `_create_fleet()` :

Here's the new `_create_fleet()` method:

invasión_extraterrestre.py

alien_invasion.py

```
def _create_fleet(self):
    """Create the fleet of aliens."""
    # Make an alien.
    alien = Alien(self)
    self.aliens.add(alien)
```

En este método, creamos una instancia de `Alien` y la añadimos al grupo que contendrá la flota. El alienígena se colocará por defecto en la zona superior izquierda de la pantalla.

In this method, we're creating one instance of `Alien` and then adding it to the group that will hold the fleet. The alien will be placed in the default upper-left area of the screen.

Para que aparezca el alienígena, tenemos que llamar al método `draw()` del grupo en `_update_screen()` :

To make the alien appear, we need to call the group's `draw()` method in `_update_screen()` :

alien_invasion.py

alien_invasion.py

```
def _update_screen(self):
    --snip--
    self.ship.blitme()
    self.aliens.draw(self.screen)

    pygame.display.flip()
```

Cuando llamas a `draw()` en un grupo, Pygame dibuja cada elemento del grupo en la posición definida por su atributo `rect`. El método `draw()` requiere un argumento: una superficie sobre la que dibujar los elementos del grupo. La Figura 13-2 muestra el primer alienígena en la pantalla.

When you call `draw()` on a group, Pygame draws each element in the group at the position defined by its `rect` attribute. The `draw()` method requires one argument: a surface on which to draw the elements from the group. Figure 13-2 shows the first alien on the screen.



DESCRIPCIÓN DE LA IMAGEN (En español):

La imagen muestra un diseño minimalista de un videojuego llamado "Invasión Alienígena". En la parte superior izquierda, se ve un gráfico que representa un alienígena clásico de estilo retro. Abajo, en el centro, se encuentra una figura que parece ser una nave espacial del jugador. El fondo es de un color gris claro, lo que enfatiza el estilo simple y retro del juego.

Figura 13-2 : Aparece el primer alienígena.

Figure 13-2 : The first alien appears.

Ahora que el primer alienígena aparece correctamente, escribiremos el código para dibujar una flota entera.

Now that the first alien appears correctly, we'll write the code to draw an entire fleet.

Construir la flota alienígena (*Building the Alien Fleet*)

Para dibujar una flota, tenemos que averiguar cómo llenar de alienígenas la parte superior de la pantalla, sin abarrotar la ventana del juego. Hay varias formas de lograr este objetivo. Lo abordaremos añadiendo alienígenas por la parte superior de la pantalla, hasta que no quede espacio para un nuevo alienígena. Entonces repetiremos este proceso, siempre que tengamos espacio vertical suficiente para añadir una nueva fila.

To draw a fleet, we need to figure out how to fill the upper portion of the screen with aliens, without overcrowding the game window. There are a number of ways to accomplish this goal. We'll approach it by adding aliens across the top of the screen, until there's no space left for a new alien. Then we'll repeat this process, as long as we have enough vertical space to add a new row.

Crear una fila de alienígenas (*Creating a Row of Aliens*)

Ahora estamos listos para generar una fila completa de alienígenas. Para crear una fila completa, primero crearemos un único alienígena, de modo que tengamos acceso a la anchura del alienígena. Colocaremos un alienígena en el lado izquierdo de la pantalla y seguiremos añadiendo alienígenas hasta que nos quedemos sin espacio:

Now we're ready to generate a full row of aliens. To make a full row, we'll first make a single alien so we have access to the alien's width. We'll place an alien on the left side of the screen and then keep adding aliens until we run out of space:

alien_invasion.py

alien_invasion.py

```
def _create_fleet(self):
    """Create the fleet of aliens."""
    # Create an alien and keep adding aliens until there's no room left.
    # Spacing between aliens is one alien width.
    alien = Alien(self)
    alien_width = alien.rect.width
```

```
❶ current_x = alien_width
❷ while current_x < (self.settings.screen_width - 2 * alien_width):
❸     new_alien = Alien(self)
❹     new_alien.x = current_x
    new_alien.rect.x = current_x
    self.aliens.add(new_alien)
❺     current_x += 2 * alien_width
```

Obtenemos la anchura del alienígena del primer alienígena que hemos creado, y luego definimos una variable llamada `current_x` ❶. Se refiere a la posición horizontal del siguiente alienígena que queremos colocar en la pantalla. Inicialmente la definimos con una anchura de alienígena, para desplazar al primer alienígena de la flota desde el borde izquierdo de la pantalla.

We get the alien's width from the first alien we created, and then define a variable called `current_x` ❶. This refers to the horizontal position of the next alien we intend to place on the screen. We initially set this to one alien width, to offset the first alien in the fleet from the left edge of the screen.

A continuación, iniciamos el bucle `while` ❷; vamos a seguir añadiendo alienígenas *while* haya espacio suficiente para colocar uno. Para determinar si hay espacio para colocar otro alienígena, compararemos `current_x` con algún valor máximo. Un primer intento de definir este bucle podría tener este aspecto:

Next, we begin the `while` loop ❷; we're going to keep adding aliens *while* there's enough room to place one. To determine whether there's room to place another alien, we'll compare `current_x` to some maximum value. A first attempt at defining this loop might look like this:

```
while current_x < self.settings.screen_width:
```

Esto parece que podría funcionar, pero colocaría al último alienígena de la fila en el extremo derecho de la pantalla. Así que añadimos un pequeño margen en el lado derecho de la pantalla. Mientras haya espacio de al menos dos anchos de alienígena en el borde derecho de la pantalla, entramos en el bucle y añadimos otro alienígena a la flota.

This seems like it might work, but it would place the final alien in the row at the far-right edge of the screen. So we add a little margin on the right side of the screen. As long as there's at least two alien widths' worth of space at the right edge of the screen, we enter the loop and add another alien to the fleet.

Siempre que haya espacio horizontal suficiente para continuar el bucle, queremos hacer dos cosas: crear un alienígena en la posición correcta y definir la posición horizontal del siguiente alienígena de la fila. Creamos un alienígena y lo asignamos a `new_alien` ❸. A continuación, fijamos la posición horizontal precisa al valor actual de `current_x` ❹. También situamos el `rect` del alienígena en este mismo valor de `x` y añadimos el nuevo alienígena al grupo `self.aliens`.

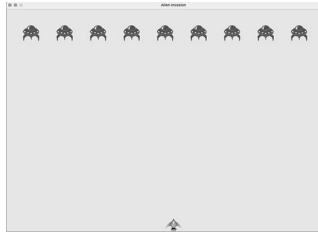
Whenever there's enough horizontal space to continue the loop, we want to do two things: create an alien at the correct position, and define the horizontal position of the next alien in the row. We create an alien and assign it to `new_alien` ❸. Then we set the precise horizontal position to the current value of `current_x` ❹. We also position the alien's `rect` at this same `x`-value, and add the new alien to the group `self.aliens`.

Por último, incrementamos el valor de `current_x` ❺. Añadimos dos anchos de alienígena a la posición horizontal, para que pase el alienígena que acabamos de añadir y para dejar también algo de espacio entre los alienígenas. Python volverá a evaluar la condición al inicio del bucle `while` y decidirá si hay espacio para otro alienígena. Cuando no quede espacio, el bucle terminará, y deberíamos tener una fila completa de alienígenas.

Finally, we increment the value of `current_x` ❺. We add two alien widths to the horizontal position, to move past the alien we just added and to leave some space between the aliens as well. Python will re-evaluate the condition at the start of the `while` loop and decide if there's room for another alien. When there's no room left, the loop will end, and we should have a full row of aliens.

Cuando ejecutes ahora *Alien Invasion*, deberías ver aparecer la primera fila de alienígenas, como en la [Figura 13-3](#).

When you run *Alien Invasion* now, you should see the first row of aliens appear, as in [Figure 13-3](#).



DESCRIPCIÓN DE LA IMAGEN (En español):

La imagen muestra un diseño que parece de un videojuego clásico. En la parte superior, se encuentra el título "Invasión Alienígena". Justo debajo, se alinean múltiples figuras de extraterrestres que se asemejan a naves o criaturas espaciales. En la parte inferior de la imagen, centrado, hay un gráfico que parece una nave espacial o una especie de defensa, posiblemente controlada por el jugador. Todo el diseño tiene un estilo monocromático y minimalista, evocando los juegos antiguos de arcade.

Figura 13-3 : La primera fila de alienígenas

Figure 13-3 : The first row of aliens

Nota (Note)

No siempre es obvio cómo construir exactamente un bucle como el que se muestra en esta sección. Una cosa buena de la programación es que tus ideas iniciales sobre cómo abordar un problema como éste no tienen por qué ser correctas. Es perfectamente razonable empezar un bucle como éste con los alienígenas situados demasiado a la derecha, y luego modificar el bucle hasta que tengas una cantidad adecuada de espacio en la pantalla.

It's not always obvious exactly how to construct a loop like the one shown in this section. One nice thing about programming is that your initial ideas for how to approach a problem like this don't have to be correct. It's perfectly reasonable to start a loop like this with the aliens positioned too far to the right, and then modify the loop until you have an appropriate amount of space on the screen.

Refactorización de `_create_fleet()` (*Refactoring `_create_fleet()`*)

Si el código que hemos escrito hasta ahora fuera todo lo que necesitamos para crear una flota, probablemente dejaríamos `_create_fleet()` tal cual. Pero tenemos más trabajo que hacer, así que vamos a limpiar un poco el método. Añadiremos un nuevo método de ayuda, `_create_alien()`, y lo llamaremos desde `_create_fleet()` :

If the code we've written so far was all we needed to create a fleet, we'd probably leave `_create_fleet()` as is. But we have more work to do, so let's clean up the method a bit. We'll add a new helper method, `_create_alien()`, and call it from `_create_fleet()` :

alien_invasion.py

alien_invasion.py

```
def _create_fleet(self):
    --snip--
    while current_x < (self.settings.screen_width - 2 * alien_width):
        self._create_alien(current_x)
        current_x += 2 * alien_width
```

```
❶ def _create_alien(self, x_position):
    """Create an alien and place it in the row."""
    new_alien = Alien(self)
    new_alien.x = x_position
    new_alien.rect.x = x_position
    self.aliens.add(new_alien)
```

El método `_create_alien()` requiere un parámetro además de `self` : el -valor `x` que especifica dónde debe colocarse el alienígena ❶ . El código en el cuerpo de `_create_alien()` es el mismo código que había en `_create_fleet()`, excepto que utilizamos el nombre de parámetro `x_position` en lugar de `current_x`. Esta refactorización facilitará la adición de nuevas filas y la creación de una flota completa.

The method `_create_alien()` requires one parameter in addition to `self` : the `x` -value that specifies where the alien should be placed ❶ . The code in the body of `_create_alien()` is the same code that was in `_create_fleet()` , except we use the parameter name `x_position` in place of `current_x` . This refactoring will make it easier to add new rows and create an entire fleet.

Añadir filas (*Adding Rows*)

Para terminar la flota, seguiremos añadiendo más filas hasta que nos quedemos sin espacio. Utilizaremos un bucle anidado: envolveremos otro bucle `while` alrededor del actual. El bucle interior colocará a los alienígenas horizontalmente en una fila centrándose en los valores `x` de los alienígenas. El bucle exterior colocará a los alienígenas verticalmente centrándose en los valores `y` . Dejaremos de añadir filas cuando nos acerquemos a la parte inferior de la pantalla, dejando espacio suficiente para la nave y algo de espacio para empezar a disparar a los alienígenas.

To finish the fleet, we'll keep adding more rows until we run out of room. We'll use a nested loop—we'll wrap another `while` loop around the current one. The inner loop will place aliens horizontally in a row by focusing on the aliens' `x` -values. The outer loop will place aliens vertically by focusing on the `y` -values. We'll stop adding rows when we get near the bottom of the screen, leaving enough space for the ship and some room to start firing at the aliens.

He aquí cómo anidar los dos bucles `while` en `_create_fleet()` :

Here's how to nest the two `while` loops in `_create_fleet()` :

```
def _create_fleet(self):
    """Create the fleet of aliens."""
    # Create an alien and keep adding aliens until there's no room left.
    # Spacing between aliens is one alien width and one alien height.
    alien = Alien(self)
    ❶ alien_width, alien_height = alien.rect.size

    ❷ current_x, current_y = alien_width, alien_height
    ❸ while current_y < (self.settings.screen_height - 3 * alien_height):
        while current_x < (self.settings.screen_width - 2 * alien_width):
```

```

❷ self._create_alien(current_x, current_y)
    current_x += 2 * alien_width

❸ # Finished a row; reset x value, and increment y value.
    current_x = alien_width
    current_y += 2 * alien_height

```

Necesitaremos saber la altura del alienígena para colocar las filas, así que cogemos la anchura y la altura del alienígena utilizando el atributo `size` de un alienígena `rect` ❶. El atributo `size` de un `rect` es una tupla que contiene su anchura y altura.

We'll need to know the alien's height in order to place rows, so we grab the alien's width and height using the `size` attribute of an alien `rect` ❶. A `rect`'s `size` attribute is a tuple containing its width and height.

A continuación, establecemos los valores iniciales `x` - `y` para la colocación del primer alienígena en la flota ❷. Lo colocamos a una anchura de alienígena hacia dentro desde la izquierda y a una altura de alienígena hacia abajo desde arriba. A continuación, definimos el bucle `while` que controla cuántas filas se colocan en la pantalla ❸. Mientras el valor `y` de la siguiente fila sea menor que la altura de la pantalla, menos tres alturas de alienígena, seguiremos añadiendo filas. (Si esto no deja el espacio adecuado, podemos ajustarlo más tarde).

Next, we set the initial `x` - and `y` -values for the placement of the first alien in the fleet ❷. We place it one alien width in from the left and one alien height down from the top. Then we define the `while` loop that controls how many rows are placed onto the screen ❸. As long as the `y` -value for the next row is less than the screen height, minus three alien heights, we'll keep adding rows. (If this doesn't leave the right amount of space, we can adjust it later.)

Llamamos a `_create_alien()`, y le pasamos el `-valor y` así como su `-posición x` ❹. Modificaremos `_create_alien()` en un momento.

We call `_create_alien()`, and pass it the `y` -value as well as its `x` -position ❹. We'll modify `_create_alien()` in a moment.

Fíjate en la sangría de las dos últimas líneas de código ⑤ . Están dentro del bucle exterior `while` , pero fuera del bucle interior `while` . Este bloque se ejecuta una vez finalizado el bucle interno; se ejecuta una vez después de crear cada fila. Después de añadir cada fila, reiniciamos el valor de `current_x` para que el primer alienígena de la siguiente fila se coloque en la misma posición que el primer alienígena de las filas anteriores. A continuación, añadimos dos alturas de alienígena al valor actual de `current_y` , para que la siguiente fila se sitúe más abajo en la pantalla. La sangría es realmente importante aquí; si no ves la flota correcta cuando ejecutes *alien_invasion.py* al final de esta sección, comprueba la sangría de todas las líneas en estos bucles anidados.

Notice the indentation of the last two lines of code ⑤ . They're inside the outer `while` loop, but outside the inner `while` loop. This block runs after the inner loop is finished; it runs once after each row is created. After each row has been added, we reset the value of `current_x` so the first alien in the next row will be placed at the same position as the first alien in the previous rows. Then we add two alien heights to the current value of `current_y` , so the next row will be placed further down the screen. Indentation is really important here; if you don't see the correct fleet when you run *alien_invasion.py* at the end of this section, check the indentation of all the lines in these nested loops.

Tenemos que modificar `_create_alien()` para fijar correctamente la posición vertical del alienígena:

We need to modify `_create_alien()` to set the vertical position of the alien correctly:

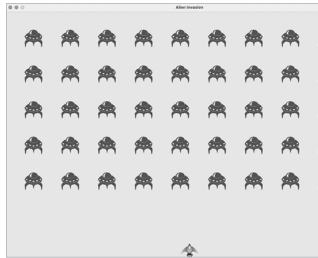
```
def _create_alien(self, x_position, y_position):
    """Create an alien and place it in the fleet."""
    new_alien = Alien(self)
    new_alien.x = x_position
    new_alien.rect.x = x_position
    new_alien.rect.y = y_position
    self.aliens.add(new_alien)
```

Modificamos la definición del método para que acepte el -valor `y` para el nuevo alienígena, y fijamos la posición vertical del `rect` en el cuerpo del método.

We modify the definition of the method to accept the `y` -value for the new alien, and we set the vertical position of the `rect` in the body of the method.

Cuando ejecutes ahora el juego, deberías ver una flota completa de alienígenas, como se muestra en la [Figura 13-4](#).

When you run the game now, you should see a full fleet of aliens, as shown in [Figure 13-4](#).



DESCRIPCIÓN DE LA
IMAGEN (En español):

En la imagen, se puede ver un diseño similar a un videojuego clásico.

La parte superior contiene el texto "Invasión Alienígena". La mayor parte de la imagen está compuesta por múltiples iconos que representan naves espaciales con formas alienígenas, organizadas en filas. Al fondo, en la parte inferior, hay una nave posicionada en el centro, apuntando hacia las figuras de arriba.

Figura 13-4 : Aparece la flota completa.

Figure 13-4 : The full fleet appears.

En la siguiente sección, ¡haremos que la flota se mueva!

In the next section, we'll make the fleet move!

Inténtalo tú mismo (*Try It Yourself*)

13-1. Estrellas: Busca la imagen de una estrella. Haz que aparezca una cuadrícula de estrellas en la pantalla.

13-1. Stars: Find an image of a star. Make a grid of stars appear on the screen.

13-2. Estrellas mejores: Puedes hacer un patrón de estrellas más realista introduciendo aleatoriedad al colocar cada estrella. Recuerda del Capítulo 9 que puedes obtener un número aleatorio de este modo:

13-2. Better Stars: You can make a more realistic star pattern by introducing randomness when you place each star. Recall from Chapter 9 that you can get a random number like this:

```
from random import randint
random_number = randint(-10, 10)
```

Este código devuelve un número entero aleatorio entre -10 y 10. Utilizando tu código en el Ejercicio 13-1, ajusta la posición de cada estrella en una cantidad aleatoria.

This code returns a random integer between -10 and 10. Using your code in Exercise 13-1, adjust each star's position by a random amount.

Hacer que la flota se mueva (*Making the Fleet Move*)

Ahora vamos a hacer que la flota de alienígenas se mueva hacia la derecha a través de la pantalla hasta que llegue al borde, y entonces haremos que baje una cantidad determinada y se mueva en la otra dirección. Continuaremos este movimiento hasta que todos los alienígenas hayan sido derribados, uno colisione con la nave o uno llegue al fondo de la pantalla. Empecemos haciendo que la flota se mueva hacia la derecha.

Now let's make the fleet of aliens move to the right across the screen until it hits the edge, and then make it drop a set amount and move in the other direction. We'll continue this movement until all aliens have been shot down, one collides with the ship, or one reaches the bottom of the screen. Let's begin by making the fleet move to the right.

Mover a los alienígenas hacia la derecha (*Moving the Aliens Right*)

Para mover a los alienígenas, utilizaremos un método `update()` en *alien.py*, que llamaremos para cada alienígena del grupo de alienígenas. Primero, añade una configuración para controlar la velocidad de cada alienígena:

To move the aliens, we'll use an `update()` method in *alien.py*, which we'll call for each alien in the group of aliens. First, add a setting to control the speed of each alien:

settings.py

settings.py

```
def __init__(self):
    --snip--
    # Alien settings
    self.alien_speed = 1.0
```

A continuación, utiliza esta configuración para implementar `update()` en *alien.py*:

Then use this setting to implement `update()` in *alien.py*:

alien.py

alien.py

```
def __init__(self, ai_game):
    """Initialize the alien and set its starting position."""
    super().__init__()
    self.screen = ai_game.screen
    self.settings = ai_game.settings
    --snip--
```

```
def update(self):
    """Move the alien to the right."""
    ❶ self.x += self.settings.alien_speed
    ❷ self.rect.x = self.x
```

Creamos un parámetro `settings` en `__init__()` para poder acceder a la velocidad del alienígena en `update()`. Cada vez que actualizamos la posición de un alienígena, lo desplazamos a la derecha de la cantidad almacenada en `alien_speed`. Seguimos la posición exacta del alienígena con el atributo `self.x`, que puede contener valores flotantes ❶. A continuación, utilizamos el valor de `self.x` para actualizar la posición del alienígena `rect` ❷.

We create a `settings` parameter in `__init__()` so we can access the alien's speed in `update()`. Each time we update an alien's position, we move it to the right by the amount stored in `alien_speed`. We track the alien's exact position with the `self.x` attribute, which can hold float values ❶. We then use the value of `self.x` to update the position of the alien's `rect` ❷.

En el bucle principal `while`, ya tenemos llamadas para actualizar las posiciones de la nave y de la bala. Ahora añadiremos una llamada para actualizar también la posición de cada alienígena:

In the main `while` loop, we already have calls to update the ship and bullet positions. Now we'll add a call to update the position of each alien as well:

alien_invasion.py

alien_invasion.py

```
while True:
    self._check_events()
    self.ship.update()
    self._update_bullets()
    self._update.aliens()
```

```
self._update_screen()
self.clock.tick(60)
```

Vamos a escribir código para gestionar el movimiento de la flota, así que creamos un nuevo método llamado `_update.aliens()` . Actualizamos las posiciones de los alienígenas después de que se hayan actualizado las balas, porque pronto comprobaremos si alguna bala ha alcanzado a algún alienígena.

We're about to write some code to manage the movement of the fleet, so we create a new method called `_update.aliens()` . We update the aliens' positions after the bullets have been updated, because we'll soon be checking to see whether any bullets hit any aliens.

El lugar donde coloques este método en el módulo no es crítico. Pero para mantener el código organizado, lo colocaré justo después de `_update.bullets()` para que coincida con el orden de las llamadas al método en el bucle `while` . Aquí tienes la primera versión de `_update.aliens()` :

Where you place this method in the module is not critical. But to keep the code organized, I'll place it just after `_update.bullets()` to match the order of method calls in the `while` loop. Here's the first version of `_update.aliens()` :

alien_invasion.py

alien_invasion.py

```
def _update.aliens(self):
    """Update the positions of all aliens in the fleet."""
    self.aliens.update()
```

Utilizamos el método `update()` en el grupo `aliens` , que llama al método `update()` de cada alienígena. Cuando ejecutes ahora *Alien Invasion* , deberías ver que la flota se mueve hacia la derecha y desaparece por el lateral de la pantalla.

We use the `update()` method on the `aliens` group, which calls each alien's `update()` method. When you run *Alien Invasion* now, you should see the fleet move right and disappear off the side of the screen.

Crear los ajustes para la dirección de la flota (*Creating Settings for Fleet Direction*)

Ahora crearemos los ajustes que harán que la flota se mueva hacia abajo en la pantalla y hacia la izquierda cuando toque el borde derecho de la pantalla. He aquí cómo implementar este comportamiento:

Now we'll create the settings that will make the fleet move down the screen and to the left when it hits the right edge of the screen. Here's how to implement this behavior:

ajustes.py

settings.py

```
# Alien settings
self.alien_speed = 1.0
self.fleet_drop_speed = 10
# fleet_direction of 1 represents right; -1 represents left.
self.fleet_direction = 1
```

El ajuste `fleet_drop_speed` controla la rapidez con la que la flota desciende por la pantalla cada vez que un alienígena alcanza cualquiera de los bordes. Es útil separar esta velocidad de la velocidad horizontal de los alienígenas para poder ajustar las dos velocidades de forma independiente.

The setting `fleet_drop_speed` controls how quickly the fleet drops down the screen each time an alien reaches either edge. It's helpful to separate this speed from the aliens' horizontal speed so you can adjust the two speeds independently.

Para implementar el ajuste `fleet_direction`, podríamos utilizar un valor de texto como `'left'` o `'right'`, pero acabaríamos con las sentencias `if - elif` comprobando la dirección de la flota. En lugar de eso, como sólo tenemos que tratar con dos direcciones, utilicemos los

valores 1 y -1 y cambiemos entre ellos cada vez que la flota cambie de dirección. (Utilizar números también tiene sentido porque moverse a la derecha implica sumar al valor de la coordenada x de cada alienígena, y moverse a la izquierda implica restar al valor de la coordenada x de cada alienígena).

To implement the setting `fleet_direction`, we could use a text value such as 'left' or 'right', but we'd end up with `if - elif` statements testing for the fleet direction. Instead, because we only have two directions to deal with, let's use the values 1 and -1 and switch between them each time the fleet changes direction. (Using numbers also makes sense because moving right involves adding to each alien's x-coordinate value, and moving left involves subtracting from each alien's x-coordinate value.)

Comprobar si un alienígena ha tocado el borde (*Checking Whether an Alien Has Hit the Edge*)

Necesitamos un método para comprobar si un alienígena está en uno de los bordes, y tenemos que modificar `update()` para permitir que cada alienígena se mueva en la dirección adecuada. Este código forma parte de la clase `Alien`:

We need a method to check whether an alien is at either edge, and we need to modify `update()` to allow each alien to move in the appropriate direction. This code is part of the `Alien` class:

alien.py

alien.py

```
def check_edges(self):
    """Return True if alien is at edge of screen."""
    screen_rect = self.screen.get_rect()
    ❶ return (self.rect.right >= screen_rect.right) or (self.rect.left <= 0)

def update(self):
    """Move the alien right or left."""
    ❷ self.x += self.settings.alien_speed * self.settings.fleet_direction
    self.rect.x = self.x
```

Podemos llamar al nuevo método `check_edges()` sobre cualquier alienígena para ver si está en el borde izquierdo o derecho. El alienígena está en el borde derecho si el atributo `right` de su `rect` es mayor o igual que el atributo `right` de la pantalla `rect`. Está en el borde izquierdo si su valor `left` es menor o igual que 0 ❶. En lugar de poner esta prueba condicional en un bloque `if`, ponemos la prueba directamente en la sentencia `return`. Este método devolverá `True` si el alienígena está en el borde derecho o izquierdo, y `False` si no está en ninguno de los bordes.

We can call the new method `check_edges()` on any alien to see whether it's at the left or right edge. The alien is at the right edge if the `right` attribute of its `rect` is greater than or equal to the `right` attribute of the screen's `rect`. It's at the left edge if its `left` value is less than or equal to 0 ❶. Rather than put this conditional test in an `if` block, we put the test directly in the `return` statement. This method will return `True` if the alien is at the right or left edge, and `False` if it is not at either edge.

Modificamos el método `update()` para permitir el movimiento hacia la izquierda o la derecha multiplicando la velocidad del alienígena por el valor de `fleet_direction` ❷. Si `fleet_direction` es 1, el valor de `alien_speed` se sumará a la posición actual del alienígena, moviéndolo hacia la derecha; si `fleet_direction` es -1, el valor se restará de la posición del alienígena, moviéndolo hacia la izquierda.

We modify the method `update()` to allow motion to the left or right by multiplying the alien's speed by the value of `fleet_direction` ❷. If `fleet_direction` is 1, the value of `alien_speed` will be added to the alien's current position, moving the alien to the right; if `fleet_direction` is -1, the value will be subtracted from the alien's position, moving the alien to the left.

Soltar la flota y cambiar de dirección (*Dropping the Fleet and Changing Direction*)

Cuando un alienígena alcanza el borde, toda la flota debe descender y cambiar de dirección. Por lo tanto, tenemos que añadir algo de código a `AlienInvasion` porque ahí es donde comprobaremos si hay algún alienígena en el borde izquierdo o derecho. Lo haremos escribiendo los métodos `_check_fleet_edges()` y `_change_fleet_direction()`, y modificando `_update.aliens()`. Pondré estos nuevos métodos después de `_create_alien()`, pero de nuevo, la ubicación de estos métodos en la clase no es crítica.

When an alien reaches the edge, the entire fleet needs to drop down and change direction. Therefore, we need to add some code to `AlienInvasion` because that's where we'll check whether any aliens are at the left or right edge. We'll make this happen by writing the methods `_check_fleet_edges()` and `_change_fleet_direction()`, and then modifying `_update.aliens()`. I'll put these new methods after `_create_alien()`, but again, the placement of these methods in the class isn't critical.

invasión_extraterrestre.py

alien_invasion.py

```
def _check_fleet_edges(self):
    """Respond appropriately if any aliens have reached an edge."""
    ❶ for alien in self.aliens.sprites():
        if alien.check_edges():
            ❷ self._change_fleet_direction()
            break

def _change_fleet_direction(self):
    """Drop the entire fleet and change the fleet's direction."""
    for alien in self.aliens.sprites():
        ❸ alien.rect.y += self.settings.fleet_drop_speed
    self.settings.fleet_direction *= -1
```

In `_check_fleet_edges()`, recorreremos la flota y llamamos a `check_edges()` en cada alienígena ❶. Si `check_edges()` devuelve `True`, sabemos que un alienígena está en un borde y que toda la flota debe cambiar de dirección; entonces llamamos a `_change_fleet_direction()` y salimos del bucle ❷. En `_change_fleet_direction()`, hacemos un bucle a través de todos los alienígenas y soltamos a cada uno utilizando la configuración `fleet_drop_speed` ❸; luego cambiamos el valor de `fleet_direction` multiplicando su valor actual por `-1`. La línea que cambia la dirección de la flota no forma parte del bucle `for`. Queremos cambiar la posición vertical de cada alienígena, pero sólo queremos cambiar la dirección de la flota una vez.

In `_check_fleet_edges()`, we loop through the fleet and call `check_edges()` on each alien ❶. If `check_edges()` returns `True`, we know an alien is at an edge and the whole fleet needs to change direction; so we call `_change_fleet_direction()` and break out of the loop ❷. In `_change_fleet_direction()`, we loop through all the aliens and drop each one using

the setting `fleet_drop_speed` ③ ; then we change the value of `fleet_direction` by multiplying its current value by `-1`. The line that changes the fleet's direction isn't part of the `for` loop. We want to change each alien's vertical position, but we only want to change the direction of the fleet once.

Estos son los cambios en `_update.aliens()` :

Here are the changes to `_update.aliens()` :

alien_invasion.py

alien_invasion.py

```
def _update.aliens(self):  
    """Check if the fleet is at an edge, then update positions."""  
    self._check_fleet_edges()  
    self.aliens.update()
```

Hemos modificado el método llamando a `_check_fleet_edges()` antes de actualizar la posición de cada alienígena.

We've modified the method by calling `_check_fleet_edges()` before updating each alien's position.

Ahora, cuando ejecutes el juego, la flota debería moverse de un lado a otro de los bordes de la pantalla y descender cada vez que toque un borde. Ahora podemos empezar a derribar alienígenas y vigilar si alguno choca contra la nave o llega a la parte inferior de la pantalla.

When you run the game now, the fleet should move back and forth between the edges of the screen and drop down every time it hits an edge. Now we can start shooting down aliens and watch for any aliens that hit the ship or reach the bottom of the screen.

Inténtalo tú mismo (Try It Yourself)

13-3. Gotas de lluvia: Busca una imagen de una gota de lluvia y crea una cuadrícula de gotas de lluvia. Haz que las gotas de lluvia caigan hacia la parte inferior de la pantalla hasta que desaparezcan.

13-3. Raindrops: Find an image of a raindrop and create a grid of raindrops. Make the raindrops fall toward the bottom of the screen until they disappear.

13-4. Lluvia constante: Modifica tu código del Ejercicio 13-3 para que cuando desaparezca una fila de gotas de lluvia de la parte inferior de la pantalla, aparezca una nueva fila en la parte superior de la pantalla y empiece a caer.

13-4. Steady Rain: Modify your code in Exercise 13-3 so when a row of raindrops disappears off the bottom of the screen, a new row appears at the top of the screen and begins to fall.

Disparando a los alienígenas (Shooting Aliens)

Hemos construido nuestra nave y una flota de alienígenas, pero cuando las balas alcanzan a los alienígenas, simplemente los atraviesan porque no estamos comprobando si hay colisiones. En programación de juegos, *collisions* se produce cuando los elementos del juego se solapan. Para que las balas derriben a los alienígenas, utilizaremos la función `sprite.groupcollide()` para buscar colisiones entre miembros de dos grupos.

We've built our ship and a fleet of aliens, but when the bullets reach the aliens, they simply pass through because we aren't checking for collisions. In game programming, *collisions* happen when game elements overlap. To make the bullets shoot down aliens, we'll use the function `sprite.groupcollide()` to look for collisions between members of two groups.

Detectar colisiones de balas (Detecting Bullet Collisions)

Queremos saber inmediatamente cuándo una bala alcanza a un alienígena, para poder hacer que desaparezca en cuanto sea alcanzado. Para ello, buscaremos colisiones

inmediatamente después de actualizar la posición de todas las balas.

We want to know right away when a bullet hits an alien so we can make an alien disappear as soon as it's hit. To do this, we'll look for collisions immediately after updating the position of all the bullets.

La función `sprite.groupcollide()` compara los `rect` s de cada elemento de un grupo con los `rect` s de cada elemento de otro grupo. En este caso, compara el `rect` de cada bala con el `rect` de cada alienígena y devuelve un diccionario que contiene las balas y los alienígenas que han colisionado. Cada clave del diccionario será una bala, y el valor correspondiente será el alienígena alcanzado. (También utilizaremos este diccionario cuando implementemos un sistema de puntuación en el Capítulo 14).

The `sprite.groupcollide()` function compares the `rect` s of each element in one group with the `rect` s of each element in another group. In this case, it compares each bullet's `rect` with each alien's `rect` and returns a dictionary containing the bullets and aliens that have collided. Each key in the dictionary will be a bullet, and the corresponding value will be the alien that was hit. (We'll also use this dictionary when we implement a scoring system in Chapter 14 .)

Añade el siguiente código al final de `_update_bullets()` para comprobar las colisiones entre balas y alienígenas:

Add the following code to the end of `_update_bullets()` to check for collisions between bullets and aliens:

alien_invasion.py

alien_invasion.py

```
def _update_bullets(self):
    """Update position of bullets and get rid of old bullets."""
    --snip--
```

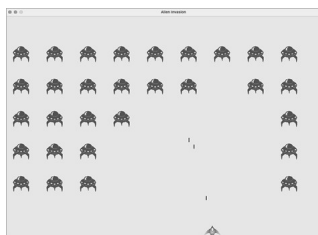
```
# Check for any bullets that have hit aliens.
# If so, get rid of the bullet and the alien.
collisions = pygame.sprite.groupcollide(
    self.bullets, self.aliens, True, True)
```

El nuevo código que hemos añadido compara las posiciones de todas las balas en `self.bullets` y de todos los alienígenas en `self.aliens`, e identifica los que se solapan. Siempre que las `rect`s de una bala y un alienígena se solapan, `groupcollide()` añade un par clave-valor al diccionario que devuelve. Los dos argumentos `True` indican a Pygame que elimine las balas y los alienígenas que hayan colisionado. (Para hacer una bala muy potente que pueda viajar hasta la parte superior de la pantalla, destruyendo a todos los alienígenas que encuentre en su camino, podrías establecer el primer argumento booleano en `False` y mantener el segundo argumento booleano en `True`. Los alienígenas alcanzados desaparecerían, pero todas las balas seguirían activas hasta que desaparecieran de la parte superior de la pantalla).

The new code we added compares the positions of all the bullets in `self.bullets` and all the aliens in `self.aliens`, and identifies any that overlap. Whenever the `rect`s of a bullet and alien overlap, `groupcollide()` adds a key-value pair to the dictionary it returns. The two `True` arguments tell Pygame to delete the bullets and aliens that have collided. (To make a high-powered bullet that can travel to the top of the screen, destroying every alien in its path, you could set the first Boolean argument to `False` and keep the second Boolean argument set to `True`. The aliens hit would disappear, but all bullets would stay active until they disappeared off the top of the screen.)

Cuando ejecutes ahora *Alien Invasion*, los alienígenas alcanzados deberían desaparecer. [La Figura 13-5](#) muestra una flota que ha sido parcialmente derribada.

When you run *Alien Invasion* now, aliens you hit should disappear. [Figure 13-5](#) shows a fleet that has been partially shot down.



DESCRIPCIÓN DE LA
IMAGEN (En español):

La imagen muestra un diseño clásico de videojuego de invasión alienígena. En la parte superior, se puede leer el título "Alien Invasion". El diseño

consiste en múltiples
filas de figuras que
representan
alienígenas,
organizadas de manera
que parecen estar
invadiendo desde el
espacio exterior. En la
parte inferior, se
observa una nave
espacial que parece ser
controlada por el
jugador. Además, hay
pequeños proyectiles
entre la nave y los
alienígenas, lo que
indica un intercambio
de disparos.

Figura 13-5 : ¡Ya podemos disparar a los alienígenas!

Figure 13-5 : We can shoot aliens!

Hacer balas más grandes para probarlas (Making Larger Bullets for Testing)

Puedes probar muchas funciones de *Alien Invasion* simplemente ejecutando el juego, pero algunas funciones son tediosas de probar en la versión normal del juego. Por ejemplo, es mucho trabajo derribar a todos los alienígenas de la pantalla varias veces para probar si tu código responde correctamente a una flota vacía.

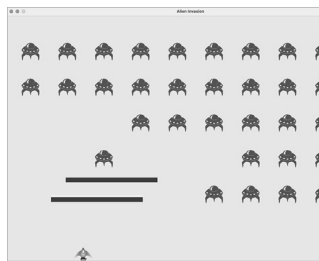
You can test many features of *Alien Invasion* simply by running the game, but some features are tedious to test in the normal version of the game. For example, it's a lot of work to shoot down every alien on the screen multiple times to test whether your code responds to an empty fleet correctly.

Para probar determinadas funciones, puedes cambiar ciertos ajustes del juego para centrarte en un área concreta. Por ejemplo, puedes encoger la pantalla para que haya menos alienígenas a los que derribar o aumentar la velocidad de las balas y darte muchas balas a la vez.

To test particular features, you can change certain game settings to focus on a particular area. For example, you might shrink the screen so there are fewer aliens to shoot down or increase the bullet speed and give yourself lots of bullets at once.

Mi cambio favorito para probar *Alien Invasion* es utilizar balas muy anchas que permanezcan activas incluso después de haber alcanzado a un alienígena (ver [Figura 13-6](#)). Prueba a poner `bullet_width` a 300, o incluso a 3.000, ¡para ver lo rápido que puedes derribar a la flota!

My favorite change for testing *Alien Invasion* is to use really wide bullets that remain active even after they've hit an alien (see [Figure 13-6](#)). Try setting `bullet_width` to 300, or even 3,000, to see how quickly you can shoot down the fleet!



DESCRIPCIÓN DE LA IMAGEN (En español):

La imagen muestra un juego clásico de invasión alienígena. En la parte superior, se puede leer el título "Invasión Alienígena". Debajo del título, hay varias filas de figuras que representan extraterrestres con forma de cangrejo, organizados en líneas. En la parte inferior izquierda, hay una figura de una nave espacial controlada por el jugador. Además, se observan dos líneas horizontales, que podrían ser barreras o protectores para la nave. El fondo es de un color gris claro, dándole un aspecto retro al juego.

Figura 13-6 : Las balas extra potentes hacen que algunos aspectos del juego sean más fáciles de probar.

Figure 13-6 : Extra-powerful bullets make some aspects of the game easier to test.

Cambios como éstos te ayudarán a probar el juego con más eficacia y posiblemente te darán ideas para dar a los jugadores poderes extra. Sólo recuerda restablecer los ajustes a la configuración normal cuando hayas terminado de probar una característica.

Changes like these will help you test the game more efficiently and possibly spark ideas for giving players bonus powers. Just remember to restore the settings to normal when you're finished testing a feature.

Repoblar la flota (*Repopulating the Fleet*)

Una característica clave de *Alien Invasion* es que los alienígenas son implacables: cada vez que se destruye la flota, debe aparecer una nueva flota.

One key feature of *Alien Invasion* is that the aliens are relentless: every time the fleet is destroyed, a new fleet should appear.

Para hacer que aparezca una nueva flota de alienígenas tras la destrucción de una flota, primero comprobamos si el grupo `aliens` está vacío. Si lo está, hacemos una llamada a `_create_fleet()`. Realizaremos esta comprobación al final de `_update_bullets()`, porque es ahí donde se destruyen los alienígenas individuales.

To make a new fleet of aliens appear after a fleet has been destroyed, we first check whether the `aliens` group is empty. If it is, we make a call to `_create_fleet()`. We'll perform this check at the end of `_update_bullets()`, because that's where individual aliens are destroyed.

`invasión_extraterrestre.py`

`alien_invasion.py`

```
def _update_bullets(self):  
    --snip--
```

```
❶ if not self.aliens:
    # Destroy existing bullets and create new fleet.
❷ self.bullets.empty()
    self._create_fleet()
```

Comprobamos si el grupo `aliens` está vacío ❶ . Un grupo vacío se evalúa en `False` , por lo que es una forma sencilla de comprobar si el grupo está vacío. Si lo está, nos deshacemos de las balas existentes utilizando el método `empty()` , que elimina todos los sprites restantes de un grupo ❷ . También llamamos a `_create_fleet()` , que vuelve a llenar la pantalla de alienígenas.

We check whether the `aliens` group is empty ❶ . An empty group evaluates to `False` , so this is a simple way to check whether the group is empty. If it is, we get rid of any existing bullets by using the `empty()` method, which removes all the remaining sprites from a group ❷ . We also call `_create_fleet()` , which fills the screen with aliens again.

Ahora aparece una nueva flota en cuanto destruyes la flota actual.

Now a new fleet appears as soon as you destroy the current fleet.

Acelerar las balas (*Speeding Up the Bullets*)

Si has intentado disparar a los alienígenas en el estado actual del juego, puede que te des cuenta de que las balas no viajan a la mejor velocidad para el juego. Pueden ser un poco demasiado lentas o un poco demasiado rápidas. En este punto, puedes modificar los ajustes para que la jugabilidad sea más interesante. Ten en cuenta que el juego se hará progresivamente más rápido, así que no lo hagas demasiado rápido al principio.

If you've tried firing at the aliens in the game's current state, you might find that the bullets aren't traveling at the best speed for gameplay. They might be a little too slow or a little too fast. At this point, you can modify the settings to make the gameplay more interesting. Keep in mind that the game is going to get progressively faster, so don't make the game too fast at the beginning.

Modificamos la velocidad de las balas ajustando el valor de `bullet_speed` en `settings.py` . En mi sistema, ajustaré el valor de `bullet_speed` a 2,5, para que las balas viajen un poco más

rápido:

We modify the speed of the bullets by adjusting the value of `bullet_speed` in *settings.py* . On my system, I'll adjust the value of `bullet_speed` to 2.5, so the bullets travel a little faster:

settings.py

settings.py

```
# Bullet settings
self.bullet_speed = 2.5
self.bullet_width = 3
--snip--
```

El mejor valor para este ajuste depende de tu experiencia en el juego, así que encuentra un valor que te funcione. También puedes ajustar otros parámetros.

The best value for this setting depends on your experience of the game, so find a value that works for you. You can adjust other settings as well.

Refactorización de `_actualizar_balas()` (*Refactoring `_update_bullets()`*)

Vamos a refactorizar `_update_bullets()` para que no haga tantas tareas diferentes. Trasladaremos el código para tratar las colisiones bala-alien a un método aparte:

Let's refactor `_update_bullets()` so it's not doing so many different tasks. We'll move the code for dealing with bullet-alien collisions to a separate method:

alien_invasion.py

alien_invasion.py

```
def _update_bullets(self):
    --snip--
    # Get rid of bullets that have disappeared.
    for bullet in self.bullets.copy():
        if bullet.rect.bottom <= 0:
            self.bullets.remove(bullet)

    self._check_bullet_alien_collisions()

def _check_bullet_alien_collisions(self):
    """Respond to bullet-alien collisions."""
    # Remove any bullets and aliens that have collided.
    collisions = pygame.sprite.groupcollide(
        self.bullets, self.aliens, True, True)

    if not self.aliens:
        # Destroy existing bullets and create new fleet.
        self.bullets.empty()
        self._create_fleet()
```

Hemos creado un nuevo método, `_check_bullet_alien_collisions()` , para buscar colisiones entre balas y alienígenas y responder adecuadamente si toda la flota ha sido destruida. Hacerlo así evita que `_update_bullets()` crezca demasiado y simplifica el desarrollo posterior.

We've created a new method, `_check_bullet_alien_collisions()` , to look for collisions between bullets and aliens and to respond appropriately if the entire fleet has been destroyed. Doing so keeps `_update_bullets()` from growing too long and simplifies further development.

Pruébalo tú mismo (*Try It Yourself*)

13-5. Tirador Lateral Parte 2: Hemos avanzado mucho desde el Ejercicio 12-6, *Sideways Shooter* . Para este ejercicio, intenta desarrollar *Sideways Shooter* hasta el mismo punto al que hemos llevado *Alien Invasion* . Añade una flota de alienígenas y haz que se muevan lateralmente hacia la nave. O bien, escribe un código que coloque a los alienígenas en posiciones aleatorias a lo largo del lado derecho de la pantalla y luego los envíe hacia la nave. Escribe también un código que haga desaparecer a los alienígenas cuando sean

alcanzados.

13-5. Sideways Shooter Part 2: We've come a long way since Exercise 12-6, *Sideways Shooter*. For this exercise, try to develop *Sideways Shooter* to the same point we've brought *Alien Invasion* to. Add a fleet of aliens, and make them move sideways toward the ship. Or, write code that places aliens at random positions along the right side of the screen and then sends them toward the ship. Also, write code that makes the aliens disappear when they're hit.

Finalizar el juego (*Ending the Game*)

¿Qué hay de divertido y desafiante en jugar a un juego que no puedes perder? Si el jugador no derriba la flota lo bastante rápido, haremos que los alienígenas destruyan la nave cuando hagan contacto. Al mismo tiempo, limitaremos el número de naves que puede utilizar el jugador, y destruiremos la nave cuando un alienígena llegue al fondo de la pantalla. El juego terminará cuando el jugador haya utilizado todas sus naves.

What's the fun and challenge in playing a game you can't lose? If the player doesn't shoot down the fleet quickly enough, we'll have the aliens destroy the ship when they make contact. At the same time, we'll limit the number of ships a player can use, and we'll destroy the ship when an alien reaches the bottom of the screen. The game will end when the player has used up all their ships.

Detección de colisiones entre naves alienígenas (*Detecting Alien-Ship Collisions*)

Empezaremos comprobando si hay colisiones entre los alienígenas y la nave, para poder responder adecuadamente cuando un alienígena choque contra ella. Comprobaremos las colisiones alienígena-nave inmediatamente después de actualizar la posición de cada alienígena en `AlienInvasion` :

We'll start by checking for collisions between aliens and the ship so we can respond appropriately when an alien hits it. We'll check for alien-ship collisions immediately after updating the position of each alien in `AlienInvasion` :

alien_invasion.py

alien_invasion.py

```
def _update.aliens(self):
    --snip--
    self.aliens.update()

    # Look for alien-ship collisions.
    ❶ if pygame.sprite.spritecollideany(self.ship, self.aliens):
    ❷     print("Ship hit!!!")
```

La función `spritecollideany()` toma dos argumentos: un `sprite` y un grupo. La función busca cualquier miembro del grupo que haya colisionado con el `sprite` y deja de recorrer el grupo en cuanto encuentra un miembro que haya colisionado con el `sprite`. En este caso, recorre el grupo `aliens` y devuelve el primer extraterrestre que encuentre que haya colisionado con `ship`.

The `spritecollideany()` function takes two arguments: a `sprite` and a group. The function looks for any member of the group that has collided with the `sprite` and stops looping through the group as soon as it finds one member that has collided with the `sprite`. Here, it loops through the group `aliens` and returns the first alien it finds that has collided with `ship`.

Si no se produce ninguna colisión, `spritecollideany()` devuelve `None` y el bloque `if` no ejecuta ❶. Si encuentra un alienígena que ha colisionado con la nave, devuelve ese alienígena y el bloque `if` se ejecuta: imprime `Ship hit!!!` ❷. Cuando un alienígena choque contra la nave, tendremos que realizar una serie de tareas: eliminar todos los alienígenas y balas restantes, recentrar la nave y crear una nueva flota. Antes de escribir código para hacer todo esto, queremos saber que nuestro enfoque para detectar colisiones alienígena-nave funciona correctamente. Escribir una llamada a `print()` es una forma sencilla de asegurarnos de que detectamos estas colisiones correctamente.

If no collisions occur, `spritecollideany()` returns `None` and the `if` block won't execute ❶. If it finds an alien that has collided with the ship, it returns that alien and the `if` block executes: it prints `Ship hit!!!` ❷. When an alien hits the ship, we'll need to do a number of tasks: delete all remaining aliens and bullets, recenter the ship, and create a new fleet.

Before we write code to do all this, we want to know that our approach to detecting alien-ship collisions works correctly. Writing a `print()` call is a simple way to ensure we're detecting these collisions properly.

Ahora, cuando ejecutes *Alien Invasion*, , debería aparecer el mensaje *Ship hit!!!* en el terminal cada vez que un alienígena choque con la nave. Cuando estés probando esta función, ajusta `fleet_drop_speed` a un valor más alto, como 50 ó 100, para que los alienígenas alcancen tu nave más rápidamente.

Now when you run *Alien Invasion*, the message *Ship hit!!!* should appear in the terminal whenever an alien runs into the ship. When you're testing this feature, set `fleet_drop_speed` to a higher value, such as 50 or 100, so the aliens reach your ship faster.

Respuesta a las colisiones entre naves alienígenas (*Responding to Alien-Ship Collisions*)

Ahora tenemos que averiguar qué ocurrirá exactamente cuando un alienígena colisione con la nave. En lugar de destruir la instancia `ship` y crear una nueva, contaremos cuántas veces ha sido golpeada la nave mediante las estadísticas de seguimiento del juego. Las estadísticas de seguimiento también serán útiles para la puntuación.

Now we need to figure out exactly what will happen when an alien collides with the ship. Instead of destroying the `ship` instance and creating a new one, we'll count how many times the ship has been hit by tracking statistics for the game. Tracking statistics will also be useful for scoring.

Escribamos una nueva clase, `GameStats` , para seguir las estadísticas del juego, y guardémosla como *game_stats.py* :

Let's write a new class, `GameStats` , to track game statistics, and let's save it as *game_stats.py* :

game_stats.py

game_stats.py

```
class GameStats:
    """Track statistics for Alien Invasion."""

    def __init__(self, ai_game):
        """Initialize statistics."""
        self.settings = ai_game.settings
        ❶ self.reset_stats()

    def reset_stats(self):
        """Initialize statistics that can change during the game."""
        self.ships_left = self.settings.ship_limit
```

Crearemos una instancia de `GameStats` para todo el tiempo que se ejecute *Alien Invasion*, pero necesitaremos restablecer algunas estadísticas cada vez que el jugador comience una nueva partida. Para ello, inicializaremos la mayoría de las estadísticas en el método `reset_stats()`, en lugar de hacerlo directamente en `__init__()`. Llamaremos a este método desde `__init__()` para que las estadísticas se establezcan correctamente cuando se cree por primera vez la instancia `GameStats` ❶. Pero también podremos llamar a `reset_stats()` cada vez que el jugador comience una nueva partida. Ahora mismo sólo tenemos una estadística, `ships_left`, cuyo valor cambiará a lo largo de la partida.

We'll make one `GameStats` instance for the entire time *Alien Invasion* is running, but we'll need to reset some statistics each time the player starts a new game. To do this, we'll initialize most of the statistics in the `reset_stats()` method, instead of directly in `__init__()`. We'll call this method from `__init__()` so the statistics are set properly when the `GameStats` instance is first created ❶. But we'll also be able to call `reset_stats()` anytime the player starts a new game. Right now we have only one statistic, `ships_left`, the value of which will change throughout the game.

El número de naves con las que comienza el jugador debe almacenarse en *settings.py* como `ship_limit` :

The number of ships the player starts with should be stored in *settings.py* as `ship_limit` :

settings.py

settings.py

```
# Ship settings
self.ship_speed = 1.5
self.ship_limit = 3
```

También tenemos que hacer algunos cambios en *alien_invasion.py* para crear una instancia de `GameStats` . En primer lugar, actualizaremos las declaraciones `import` de la parte superior del archivo:

We also need to make a few changes in *alien_invasion.py* to create an instance of `GameStats` . First, we'll update the `import` statements at the top of the file:

alien_invasion.py

alien_invasion.py

```
import sys
from time import sleep

import pygame

from settings import Settings
from game_stats import GameStats
from ship import Ship
--snip--
```

Importamos la función `sleep()` del módulo `time` de la biblioteca estándar de Python, para poder pausar el juego un momento cuando la nave sea alcanzada. También importamos `GameStats` .

We import the `sleep()` function from the `time` module in the Python standard library, so we can pause the game for a moment when the ship is hit. We also import `GameStats` .

Crearemos una instancia de `GameStats` en `__init__()` :

We'll create an instance of `GameStats` in `__init__()` :

alien_invasion.py

alien_invasion.py

```
def __init__(self):
    --snip--
    self.screen = pygame.display.set_mode(
        (self.settings.screen_width, self.settings.screen_height))
    pygame.display.set_caption("Alien Invasion")

    # Create an instance to store game statistics.
    self.stats = GameStats(self)

    self.ship = Ship(self)
    --snip--
```

Creamos la instancia después de crear la ventana del juego, pero antes de definir otros elementos del juego, como la nave.

We make the instance after creating the game window but before defining other game elements, such as the ship.

Cuando un alienígena impacte contra la nave, restaremos 1 al número de naves restantes, destruiremos todos los alienígenas y balas existentes, crearemos una nueva flota y recolocaremos la nave en el centro de la pantalla. También pausaremos el juego un momento para que el jugador pueda darse cuenta de la colisión y reagruparse antes de que aparezca una nueva flota.

When an alien hits the ship, we'll subtract 1 from the number of ships left, destroy all existing aliens and bullets, create a new fleet, and reposition the ship in the middle of the screen. We'll also pause the game for a moment so the player can notice the collision and regroup before a new fleet appears.

Pongamos la mayor parte de este código en un nuevo método llamado `_ship_hit()` . Llamaremos a este método desde `_update_aliens()` cuando un alien alcance la nave:

Let's put most of this code in a new method called `_ship_hit()` . We'll call this method from `_update_aliens()` when an alien hits the ship:

alien_invasion.py

alien_invasion.py

```
def _ship_hit(self):
    """Respond to the ship being hit by an alien."""
    # Decrement ships_left.
    ❶ self.stats.ships_left -= 1

    # Get rid of any remaining bullets and aliens.
    ❷ self.bullets.empty()
    self.aliens.empty()

    # Create a new fleet and center the ship.
    ❸ self._create_fleet()
    self.ship.center_ship()

    # Pause.
    ❹ sleep(0.5)
```

El nuevo método `_ship_hit()` coordina la respuesta cuando un alienígena golpea una nave. Dentro de `_ship_hit()` , el número de naves restantes se reduce en 1 ❶ , tras lo cual vaciamos los grupos `bullets` y `aliens` ❷ .

The new method `_ship_hit()` coordinates the response when an alien hits a ship. Inside `_ship_hit()`, the number of ships left is reduced by 1 ❶, after which we empty the groups `bullets` and `aliens` ❷.

A continuación, creamos una nueva flota y centramos la nave ❸. (En un momento añadiremos el método `center_ship()` a `Ship`.) A continuación, añadimos una pausa después de que se hayan realizado las actualizaciones de todos los elementos del juego, pero antes de que se haya dibujado ningún cambio en la pantalla, para que el jugador pueda ver que su nave ha sido golpeada ❹. La llamada a `sleep()` pausa la ejecución del programa durante medio segundo, el tiempo suficiente para que el jugador vea que el alienígena ha golpeado la nave. Cuando la función `sleep()` termina, la ejecución del código pasa al método `_update_screen()`, que dibuja la nueva flota en la pantalla.

Next, we create a new fleet and center the ship ❸. (We'll add the method `center_ship()` to `Ship` in a moment.) Then we add a pause after the updates have been made to all the game elements but before any changes have been drawn to the screen, so the player can see that their ship has been hit ❹. The `sleep()` call pauses program execution for half a second, long enough for the player to see that the alien has hit the ship. When the `sleep()` function ends, code execution moves on to the `_update_screen()` method, which draws the new fleet to the screen.

En `_update_aliens()`, sustituimos la llamada a `print()` por una llamada a `_ship_hit()` cuando un alienígena golpea la nave:

In `_update_aliens()`, we replace the `print()` call with a call to `_ship_hit()` when an alien hits the ship:

alien_invasion.py

alien_invasion.py

```
def _update.aliens(self):
    --snip--
    if pygame.sprite.spritecollideany(self.ship, self.aliens):
        self._ship_hit()
```

Aquí está el nuevo método `center_ship()` , que pertenece a *ship.py* :

Here's the new method `center_ship()` , which belongs in *ship.py* :

nave.py

ship.py

```
def center_ship(self):
    """Center the ship on the screen."""
    self.rect.midbottom = self.screen_rect.midbottom
    self.x = float(self.rect.x)
```

Centramos la nave del mismo modo que en `__init__()` . Después de centrarla, reseteamos el atributo `self.x` , que nos permite seguir la posición exacta de la nave.

We center the ship the same way we did in `__init__()` . After centering it, we reset the `self.x` attribute, which allows us to track the ship's exact position.

Nota (Note)

Observa que nunca hacemos más de una nave; sólo hacemos una instancia de nave para todo el juego y la volvemos a centrar siempre que la nave haya sido golpeada. La estadística `ships_left` nos indicará cuándo el jugador se ha quedado sin naves.

Notice that we never make more than one ship; we make only one ship instance for the whole game and recenter it whenever the ship has been hit. The statistic `ships_left` will tell us when the player has run out of ships.

Ejecuta el juego, dispara a unos cuantos alienígenas y deja que un alienígena golpee la nave. El juego debería hacer una pausa, y debería aparecer una nueva flota con la nave centrada de nuevo en la parte inferior de la pantalla.

Run the game, shoot a few aliens, and let an alien hit the ship. The game should pause, and a new fleet should appear with the ship centered at the bottom of the screen again.

Alienígenas que llegan al fondo de la pantalla (*Aliens That Reach the Bottom of the Screen*)

Si un alienígena llega a la parte inferior de la pantalla, haremos que el juego responda del mismo modo que cuando un alienígena golpea la nave. Para comprobar cuándo ocurre esto, añade un nuevo método en *alien_invasion.py* :

If an alien reaches the bottom of the screen, we'll have the game respond the same way it does when an alien hits the ship. To check when this happens, add a new method in *alien_invasion.py* :

alien_invasion.py

alien_invasion.py

```
def _check.aliens_bottom(self):
    """Check if any aliens have reached the bottom of the screen."""
    for alien in self.aliens.sprites():
        ❶ if alien.rect.bottom >= self.settings.screen_height:
            # Treat this the same as if the ship got hit.
            self._ship_hit()
            break
```

El método `_check.aliens_bottom()` comprueba si algún alienígena ha llegado al fondo de la pantalla. Un alienígena llega al fondo cuando su valor `rect.bottom` es mayor o igual que la altura de la pantalla ❶. Si un alienígena llega al fondo, llamamos a `_ship_hit()`. Si un alienígena llega al fondo, no es necesario comprobar el resto, así que salimos del bucle

después de llamar a `_ship_hit()` .

The method `_check.aliens.bottom()` checks whether any aliens have reached the bottom of the screen. An alien reaches the bottom when its `rect.bottom` value is greater than or equal to the screen's height ❶ . If an alien reaches the bottom, we call `_ship_hit()` . If one alien hits the bottom, there's no need to check the rest, so we break out of the loop after calling `_ship_hit()` .

Llamaremos a este método desde `_update.aliens()` :

We'll call this method from `_update.aliens()` :

alien_invasion.py

alien_invasion.py

```
def _update.aliens(self):
    --snip--
    # Look for alien-ship collisions.
    if pygame.sprite.spritecollideany(self.ship, self.aliens):
        self._ship_hit()

    # Look for aliens hitting the bottom of the screen.
    self._check.aliens.bottom()
```

Llamamos a `_check.aliens.bottom()` después de actualizar las posiciones de todos los alienígenas y después de buscar colisiones alienígena-nave. Ahora aparecerá una nueva flota cada vez que la nave sea alcanzada por un alienígena o que un alienígena llegue al fondo de la pantalla.

We call `_check.aliens.bottom()` after updating the positions of all the aliens and after looking for alien-ship collisions. Now a new fleet will appear every time the ship is hit by an alien or an alien reaches the bottom of the screen.

¡Game Over! (Game Over!)

Alien Invasion parece más completo ahora, pero el juego nunca termina. El valor de `ships_left` sólo se vuelve cada vez más negativo. Añadamos una bandera `game_active`, para que podamos terminar el juego cuando el jugador se quede sin naves. Estableceremos esta bandera al final del método `__init__()` en `AlienInvasion`:

Alien Invasion feels more complete now, but the game never ends. The value of `ships_left` just grows increasingly negative. Let's add a `game_active` flag, so we can end the game when the player runs out of ships. We'll set this flag at the end of the `__init__()` method in `AlienInvasion`:

alien_invasion.py

alien_invasion.py

```
def __init__(self):
    --snip--
    # Start Alien Invasion in an active state.
    self.game_active = True
```

Ahora añadiremos código a `_ship_hit()` que establezca `game_active` en `False` cuando el jugador haya agotado todas sus naves:

Now we add code to `_ship_hit()` that sets `game_active` to `False` when the player has used up all their ships:

alien_invasion.py

alien_invasion.py

```
def _ship_hit(self):
    """Respond to ship being hit by alien."""
```

```
if self.stats.ships_left > 0:
    # Decrement ships_left.
    self.stats.ships_left -= 1
    --snip--
    # Pause.
    sleep(0.5)
else:
    self.game_active = False
```

La mayor parte de `_ship_hit()` no se modifica. Hemos trasladado todo el código existente a un bloque `if`, que comprueba que al jugador le queda al menos una nave. Si es así, creamos una nueva flota, hacemos una pausa y seguimos adelante. Si al jugador no le queda ninguna nave, ponemos `game_active` en `False`.

Most of `_ship_hit()` is unchanged. We've moved all the existing code into an `if` block, which tests to make sure the player has at least one ship remaining. If so, we create a new fleet, pause, and move on. If the player has no ships left, we set `game_active` to `False`.

Identificar cuándo deben ejecutarse las partes del juego (*Identifying When Parts of the Game Should Run*)

Tenemos que identificar las partes del juego que deben ejecutarse siempre y las que sólo deben ejecutarse cuando el juego está activo:

We need to identify the parts of the game that should always run and the parts that should run only when the game is active:

alien_invasion.py

alien_invasion.py

```
def run_game(self):
    """Start the main loop for the game."""
    while True:
        self._check_events()
```

```
if self.game_active:
    self.ship.update()
    self._update_bullets()
    self._update.aliens()

self._update_screen()
self.clock.tick(60)
```

En el bucle principal, siempre tenemos que llamar a `_check_events()` , aunque el juego esté inactivo. Por ejemplo, seguimos necesitando saber si el usuario pulsa Q para salir del juego o si hace clic en el botón para cerrar la ventana. También seguimos actualizando la pantalla para poder hacer cambios en ella mientras esperamos a ver si el jugador decide empezar una nueva partida. El resto de las llamadas a funciones sólo tienen que ocurrir cuando el juego está activo, porque cuando el juego está inactivo, no necesitamos actualizar las posiciones de los elementos del juego.

In the main loop, we always need to call `_check_events()` , even if the game is inactive. For example, we still need to know if the user presses Q to quit the game or clicks the button to close the window. We also continue updating the screen so we can make changes to the screen while waiting to see whether the player chooses to start a new game. The rest of the function calls need to happen only when the game is active, because when the game is inactive, we don't need to update the positions of game elements.

Ahora, cuando juegues a *Alien Invasion* , el juego debería congelarse cuando hayas utilizado todas tus naves.

Now when you play *Alien Invasion* , the game should freeze when you've used up all your ships.

Pruébalo tú mismo (*Try It Yourself*)

13-6. Fin del juego: En *Sideways Shooter* , lleva la cuenta del número de veces que la nave es alcanzada y del número de veces que un alien es alcanzado por la nave. Decide una condición apropiada para terminar la partida, y detén el juego cuando se produzca esta situación.

13-6. Game Over: In *Sideways Shooter*, keep track of the number of times the ship is hit and the number of times an alien is hit by the ship. Decide on an appropriate condition for ending the game, and stop the game when this situation occurs.

Resumen (Summary)

En este capítulo has aprendido a añadir un gran número de elementos idénticos a un juego creando una flota de alienígenas. Utilizaste bucles anidados para crear una cuadrícula de elementos, e hiciste que un gran conjunto de elementos del juego se movieran llamando al método `update()` de cada elemento. Aprendiste a controlar la dirección de los objetos en la pantalla y a responder a situaciones concretas, como cuando la flota alcanza el borde de la pantalla. Detectaste y respondiste a las colisiones cuando las balas golpean a los alienígenas y los alienígenas golpean a la nave. También aprendiste a seguir las estadísticas de una partida y a utilizar una bandera `game_active` para determinar cuándo termina la partida.

In this chapter, you learned how to add a large number of identical elements to a game by creating a fleet of aliens. You used nested loops to create a grid of elements, and you made a large set of game elements move by calling each element's `update()` method. You learned to control the direction of objects on the screen and to respond to specific situations, such as when the fleet reaches the edge of the screen. You detected and responded to collisions when bullets hit aliens and aliens hit the ship. You also learned how to track statistics in a game and use a `game_active` flag to determine when the game is over.

En el próximo y último capítulo de este proyecto, añadiremos un botón Jugar para que el jugador pueda elegir cuándo empezar su primera partida y si quiere volver a jugar cuando termine la partida. Aceleraremos el juego cada vez que el jugador derribe toda la flota, y añadiremos un sistema de puntuación. ¡El resultado final será un juego totalmente jugable!

In the next and final chapter of this project, we'll add a Play button so the player can choose when to start their first game and whether to play again when the game ends. We'll speed up the game each time the player shoots down the entire fleet, and we'll add a scoring system. The final result will be a fully playable game!

Capítulo 14: Puntuación (Chapter 14: Scoring)

14

Puntuación (14 Scoring)



DESCRIPCIÓN DE LA
IMAGEN (En español):

La imagen muestra una ilustración en blanco y negro de una serpiente con gafas, que está enrollada alrededor de un motor a reacción. La serpiente tiene una expresión determinada y parece estar en movimiento, transmitiendo un sentido de velocidad y energía. El diseño es estilizado y dinámico, destacando tanto la figura de la serpiente como el motor. No hay texto presente en esta imagen.

En este capítulo, terminaremos de construir *Alien Invasion* . Añadiremos un botón Reproducir para iniciar el juego cuando se solicite y para reiniciarlo cuando termine. También cambiaremos el juego para que se acelere cuando el jugador suba de nivel, e implementaremos un sistema de puntuación. Al final del capítulo, sabrás lo suficiente para empezar a escribir juegos que aumenten su dificultad a medida que el jugador progresa y que incorporen sistemas completos de puntuación.

In this chapter, we'll finish building *Alien Invasion* . We'll add a Play button to start the game on demand and to restart the game once it ends. We'll also change the game so it speeds up when the player moves up a level, and we'll implement a scoring system. By the end of the chapter, you'll know enough to start writing games that increase in difficulty as a player progresses and that feature complete scoring systems.

Añadir el botón Jugar (Adding the Play Button)

En esta sección, añadiremos un botón Jugar que aparecerá antes de que empiece una partida y reaparecerá cuando ésta termine para que el jugador pueda volver a jugar.

In this section, we'll add a Play button that appears before a game begins and reappears when the game ends so the player can play again.

Ahora mismo, el juego comienza en cuanto ejecutas *alien_invasion.py*. Empecemos el juego en un estado inactivo y luego pidamos al jugador que pulse el botón Jugar para empezar. Para ello, modifica el método `__init__()` de `AlienInvasion` :

Right now, the game begins as soon as you run *alien_invasion.py*. Let's start the game in an inactive state and then prompt the player to click a Play button to begin. To do this, modify the `__init__()` method of `AlienInvasion` :

alien_invasion.py

alien_invasion.py

```
def __init__(self):
    """Initialize the game, and create game resources."""
    pygame.init()
    --snip--

    # Start Alien Invasion in an inactive state.
    self.game_active = False
```

Ahora el juego debería comenzar en un estado inactivo, sin forma de que el jugador lo inicie hasta que hagamos un botón Reproducir.

Now the game should start in an inactive state, with no way for the player to start it until we make a Play button.

Crear una clase de botón (*Creating a Button Class*)

Como Pygame no tiene un método incorporado para crear botones, escribiremos una clase `Button` para crear un rectángulo relleno con una etiqueta. Puedes utilizar este código para crear cualquier botón en un juego. Aquí tienes la primera parte de la clase `Button` ; guárdala como *button.py* :

Because Pygame doesn't have a built-in method for making buttons, we'll write a `Button` class to create a filled rectangle with a label. You can use this code to make any button in a game. Here's the first part of the `Button` class; save it as *button.py* :

boton.py

button.py

```
import pygame.font
```

```
class Button:
```

```
    """A class to build buttons for the game."""
```

```
❶ def __init__(self, ai_game, msg):
```

```
    """Initialize button attributes."""
```

```
    self.screen = ai_game.screen
```

```
    self.screen_rect = self.screen.get_rect()
```

```
    # Set the dimensions and properties of the button.
```

```
❷ self.width, self.height = 200, 50
```

```
    self.button_color = (0, 135, 0)
```

```
    self.text_color = (255, 255, 255)
```

```
❸ self.font = pygame.font.SysFont(None, 48)
```

```
    # Build the button's rect object and center it.
```

```
❹ self.rect = pygame.Rect(0, 0, self.width, self.height)
```

```
    self.rect.center = self.screen_rect.center
```

```
    # The button message needs to be prepped only once.
```

```
❺ self._prep_msg(msg)
```

En primer lugar, importamos el módulo `pygame.font` , que permite a Pygame renderizar texto en la pantalla. El método `__init__()` toma los parámetros `self` , el objeto `ai_game` , y `msg` , que contiene el texto del botón ❶ . Establecemos las dimensiones del botón ❷ , establecemos `button_color` para colorear el objeto `rect` del botón en verde oscuro, y establecemos `text_color` para renderizar el texto en blanco.

First, we import the `pygame.font` module, which lets Pygame render text to the screen. The `__init__()` method takes the parameters `self` , the `ai_game` object, and `msg` , which contains the button's text ❶ . We set the button dimensions ❷ , set `button_color` to color the button's `rect` object dark green, and set `text_color` to render the text in white.

A continuación, preparamos un atributo `font` para representar el texto ❸ . El argumento `None` indica a Pygame que utilice la fuente por defecto, y `48` especifica el tamaño del texto. Para centrar el botón en la pantalla, creamos un `rect` para el botón ❹ y establecemos su atributo `center` para que coincida con el de la pantalla.

Next, we prepare a `font` attribute for rendering text ❸ . The `None` argument tells Pygame to use the default font, and `48` specifies the size of the text. To center the button on the screen, we create a `rect` for the button ❹ and set its `center` attribute to match that of the screen.

Pygame trabaja con texto representando la cadena que quieres mostrar como una imagen. Por último, llamamos a `_prep_msg()` para que se encargue de esta renderización ❺ .

Pygame works with text by rendering the string you want to display as an image. Finally, we call `_prep_msg()` to handle this rendering ❺ .

Este es el código de `_prep_msg()` :

Here's the code for `_prep_msg()` :

boton.py

button.py

```
def _prep_msg(self, msg):
    """Turn msg into a rendered image and center text on the button."""
    ❶ self.msg_image = self.font.render(msg, True, self.text_color,
        self.button_color)
    ❷ self.msg_image_rect = self.msg_image.get_rect()
    self.msg_image_rect.center = self.rect.center
```

El método `_prep_msg()` necesita un parámetro `self` y el texto a representar como imagen (`msg`). La llamada a `font.render()` convierte el texto almacenado en `msg` en una imagen, que luego almacenamos en `self.msg_image` ❶ . El método `font.render()` también toma un valor booleano para activar o desactivar el antialiasing (*antialiasing* hace que los bordes del texto sean más suaves). Los argumentos restantes son el color de fuente y el color de fondo especificados. Establecemos el antialiasing en `True` y el fondo del texto en el mismo color que el botón. (Si no incluyes un color de fondo, Pygame intentará renderizar la fuente con un fondo transparente).

The `_prep_msg()` method needs a `self` parameter and the text to be rendered as an image (`msg`). The call to `font.render()` turns the text stored in `msg` into an image, which we then store in `self.msg_image` ❶ . The `font.render()` method also takes a Boolean value to turn antialiasing on or off (*antialiasing* makes the edges of the text smoother). The remaining arguments are the specified font color and background color. We set antialiasing to `True` and set the text background to the same color as the button. (If you don't include a background color, Pygame will try to render the font with a transparent background.)

Centramos la imagen del texto en el botón creando un `rect` a partir de la imagen y ajustando su atributo `center` para que coincida con el del botón ❷ .

We center the text image on the button by creating a `rect` from the image and setting its `center` attribute to match that of the button ❷ .

Por último, creamos un método `draw_button()` al que podemos llamar para mostrar el botón en pantalla:

Finally, we create a `draw_button()` method that we can call to display the button onscreen:

boton.py

button.py

```
def draw_button(self):
    """Draw blank button and then draw message."""
    self.screen.fill(self.button_color, self.rect)
    self.screen.blit(self.msg_image, self.msg_image_rect)
```

Llamamos a `screen.fill()` para dibujar la parte rectangular del botón. Luego llamamos a `screen.blit()` para que dibuje la imagen del texto en la pantalla, pasándole una imagen y el objeto `rect` asociado a la imagen. Con esto completamos la clase `Button` .

We call `screen.fill()` to draw the rectangular portion of the button. Then we call `screen.blit()` to draw the text image to the screen, passing it an image and the `rect` object associated with the image. This completes the `Button` class.

Dibujar el botón en la pantalla (*Drawing the Button to the Screen*)

Utilizaremos la clase `Button` para crear un botón Reproducir en `AlienInvasion` . En primer lugar, actualizaremos las sentencias `import` :

We'll use the `Button` class to create a Play button in `AlienInvasion` . First, we'll update the `import` statements:

alien_invasion.py

alien_invasion.py

--snip--

```
from game_stats import GameStats
from button import Button
```

Como sólo necesitamos un botón Reproducir, crearemos el botón en el método `__init__()` de `AlienInvasion` . Podemos colocar este código justo al final de `__init__()` :

Because we need only one Play button, we'll create the button in the `__init__()` method of `AlienInvasion` . We can place this code at the very end of `__init__()` :

alien_invasion.py

alien_invasion.py

```
def __init__(self):
    --snip--
    self.game_active = False

    # Make the Play button.
    self.play_button = Button(self, "Play")
```

Este código crea una instancia de `Button` con la etiqueta `Play` , pero no dibuja el botón en la pantalla. Para ello, llamaremos al método `draw_button()` del botón en `_update_screen()` :

This code creates an instance of `Button` with the label `Play` , but it doesn't draw the button to the screen. To do this, we'll call the button's `draw_button()` method in `_update_screen()` :

alien_invasion.py

alien_invasion.py

```
def _update_screen(self):
    --snip--
    self.aliens.draw(self.screen)

    # Draw the play button if the game is inactive.
    if not self.game_active:
        self.play_button.draw_button()

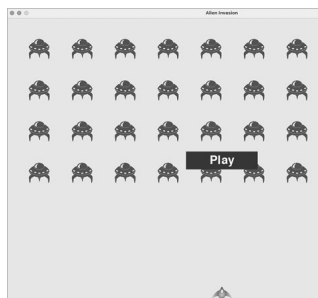
    pygame.display.flip()
```

Para que el botón Reproducir sea visible por encima de todos los demás elementos de la pantalla, lo dibujamos después de que se hayan dibujado todos los demás elementos, pero antes de pasar a una nueva pantalla. Lo incluimos en un bloque `if`, para que el botón sólo aparezca cuando el juego esté inactivo.

To make the Play button visible above all other elements on the screen, we draw it after all the other elements have been drawn but before flipping to a new screen. We include it in an `if` block, so the button only appears when the game is inactive.

Ahora, cuando ejecutes *Alien Invasion*, deberías ver un botón Jugar en el centro de la pantalla, como se muestra en la [Figura 14-1](#).

Now when you run *Alien Invasion*, you should see a Play button in the center of the screen, as shown in [Figure 14-1](#).



DESCRIPCIÓN DE LA
IMAGEN (En español):

Invasión Alienígena
Jugar

Figura 14-1 : Aparece un botón Jugar cuando el juego está inactivo.

Figure 14-1 : A Play button appears when the game is inactive.

Iniciar el juego (*Starting the Game*)

Para iniciar una nueva partida cuando el jugador haga clic en Jugar, añade el siguiente bloque `elif` al final de `_check_events()` para controlar los eventos del ratón sobre el botón:

To start a new game when the player clicks Play, add the following `elif` block to the end of `_check_events()` to monitor mouse events over the button:

alien_invasion.py

alien_invasion.py

```
def _check_events(self):
    """Respond to keypresses and mouse events."""
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            --snip--
        ❶ elif event.type == pygame.MOUSEBUTTONDOWN:
        ❷     mouse_pos = pygame.mouse.get_pos()
        ❸     self._check_play_button(mouse_pos)
```

Pygame detecta un evento `MOUSEBUTTONDOWN` cuando el jugador hace clic en cualquier parte de la pantalla ❶, pero queremos restringir nuestro juego para que responda a los clics del ratón sólo sobre el botón Jugar. Para ello, utilizamos `pygame.mouse.get_pos()`, que devuelve una tupla con las coordenadas *x* - *y* y *y* - *x* del cursor del ratón cuando se pulsa el botón del ratón ❷. Enviamos estos valores al nuevo método `_check_play_button()` ❸.

Pygame detects a `MOUSEBUTTONDOWN` event when the player clicks anywhere on the screen ❶, but we want to restrict our game to respond to mouse clicks only on the Play button. To accomplish this, we use `pygame.mouse.get_pos()`, which returns a tuple containing the mouse cursor's *x* - and *y* -coordinates when the mouse button is clicked ❷. We send these values to the new method `_check_play_button()` ❸.

Aquí está `_check_play_button()` , que decidí colocar después de `_check_events()` :

Here's `_check_play_button()` , which I chose to place after `_check_events()` :

invasión_extraterrestre.py

alien_invasion.py

```
def _check_play_button(self, mouse_pos):
    """Start a new game when the player clicks Play."""
    ❶ if self.play_button.rect.collidepoint(mouse_pos):
        self.game_active = True
```

Utilizamos el método `rect.collidepoint()` para comprobar si el punto del clic del ratón se solapa con la región definida por el botón Reproducir de `rect` ❶ . Si es así, establecemos `game_active` en `True` , ¡y comienza el juego!

We use the `rect` method `collidepoint()` to check whether the point of the mouse click overlaps the region defined by the Play button's `rect` ❶ . If so, we set `game_active` to `True` , and the game begins!

Llegados a este punto, deberías poder iniciar y jugar una partida completa. Cuando el juego termine, el valor de `game_active` debería convertirse en `False` y el botón Jugar debería reaparecer.

At this point, you should be able to start and play a full game. When the game ends, the value of `game_active` should become `False` and the Play button should reappear.

Reiniciar el juego (*Resetting the Game*)

El código del botón Jugar que acabamos de escribir funciona la primera vez que el jugador pulsa Jugar. Pero no funciona después de que termine la primera partida, porque las condiciones que hicieron que terminara la partida no se han reiniciado.

The Play button code we just wrote works the first time the player clicks Play. But it doesn't work after the first game ends, because the conditions that caused the game to end haven't been reset.

Para reiniciar la partida cada vez que el jugador haga clic en Jugar, tenemos que reiniciar las estadísticas de la partida, eliminar los antiguos alienígenas y balas, construir una nueva flota y centrar la nave, como se muestra aquí:

To reset the game each time the player clicks Play, we need to reset the game statistics, clear out the old aliens and bullets, build a new fleet, and center the ship, as shown here:

alien_invasion.py

alien_invasion.py

```
def _check_play_button(self, mouse_pos):
    """Start a new game when the player clicks Play."""
    if self.play_button.rect.collidepoint(mouse_pos):
        # Reset the game statistics.
        ❶ self.stats.reset_stats()
        self.game_active = True

        # Get rid of any remaining bullets and aliens.
        ❷ self.bullets.empty()
        self.aliens.empty()

        # Create a new fleet and center the ship.
        ❸ self._create_fleet()
        self.ship.center_ship()
```

Reiniciamos las estadísticas del juego ❶ , lo que proporciona al jugador tres naves nuevas. Luego establecemos `game_active` en `True` para que el juego comience en cuanto termine de ejecutarse el código de esta función. Vaciamos los grupos `aliens` y `bullets` ❷ , y luego creamos una nueva flota y centramos la nave ❸ .

We reset the game statistics ❶ , which gives the player three new ships. Then we set `game_active` to `True` so the game will begin as soon as the code in this function finishes running. We empty the `aliens` and `bullets` groups ❷ , and then we create a new fleet and center the ship ❸ .

Ahora el juego se reiniciará correctamente cada vez que pulses Jugar, ¡lo que te permitirá jugarlo tantas veces como quieras!

Now the game will reset properly each time you click Play, allowing you to play it as many times as you want!

Desactivar el botón Jugar (*Deactivating the Play Button*)

Un problema de nuestro botón Jugar es que la región del botón en la pantalla seguirá respondiendo a los clics aunque el botón Jugar no esté visible. Si haces clic en la zona del botón Jugar por accidente después de que empiece una partida, ¡el juego se reiniciará!

One issue with our Play button is that the button region on the screen will continue to respond to clicks even when the Play button isn't visible. If you click the Play button area by accident after a game begins, the game will restart!

Para solucionarlo, configura el juego para que sólo se inicie cuando `game_active` sea `False` :

To fix this, set the game to start only when `game_active` is `False` :

alien_invasion.py

alien_invasion.py

```
def _check_play_button(self, mouse_pos):
    """Start a new game when the player clicks Play."""
    ❶ button_clicked = self.play_button.rect.collidepoint(mouse_pos)
    ❷ if button_clicked and not self.game_active:
        # Reset the game statistics.
        self.stats.reset_stats()
        --snip--
```

La bandera `button_clicked` almacena un valor `True` o `False` ❶, y el juego se reiniciará sólo si se pulsa Jugar *and* el juego no está activo en ese momento ❷. Para probar este comportamiento, inicia una nueva partida y pulsa repetidamente donde debería estar el botón Jugar. Si todo funciona como se espera, hacer clic en la zona del botón Jugar no debería tener ningún efecto en el juego.

The flag `button_clicked` stores a `True` or `False` value ❶, and the game will restart only if Play is clicked *and* the game is not currently active ❷. To test this behavior, start a new game and repeatedly click where the Play button should be. If everything works as expected, clicking the Play button area should have no effect on the gameplay.

Ocultar el cursor del ratón (*Hiding the Mouse Cursor*)

Queremos que el cursor del ratón esté visible cuando el juego está inactivo, pero una vez que comienza el juego, simplemente estorba. Para solucionar esto, lo haremos invisible cuando el juego se active. Podemos hacerlo al final del bloque `if` en `_check_play_button()`:

We want the mouse cursor to be visible when the game is inactive, but once play begins, it just gets in the way. To fix this, we'll make it invisible when the game becomes active. We can do this at the end of the `if` block in `_check_play_button()`:

alien_invasion.py

alien_invasion.py

```
def _check_play_button(self, mouse_pos):
    """Start a new game when the player clicks Play."""
```

```
button_clicked = self.play_button.rect.collidepoint(mouse_pos)
if button_clicked and not self.game_active:
    --snip--
    # Hide the mouse cursor.
    pygame.mouse.set_visible(False)
```

Si pasas `False` a `set_visible()` , le dirás a Pygame que oculte el cursor cuando el ratón esté sobre la ventana del juego.

Passing `False` to `set_visible()` tells Pygame to hide the cursor when the mouse is over the game window.

Haremos que el cursor reaparezca cuando termine la partida para que el jugador pueda volver a pulsar Jugar para empezar una nueva partida. Aquí tienes el código para hacerlo:

We'll make the cursor reappear once the game ends so the player can click Play again to begin a new game. Here's the code to do that:

invasión_extraterrestre.py

alien_invasion.py

```
def _ship_hit(self):
    """Respond to ship being hit by alien."""
    if self.stats.ships_left > 0:
        --snip--
    else:
        self.game_active = False
        pygame.mouse.set_visible(True)
```

Hacemos que el cursor vuelva a ser visible en cuanto el juego se vuelve inactivo, lo que ocurre en `_ship_hit()` . Prestar atención a detalles como éste hace que tu juego tenga un aspecto más profesional y permite que el jugador se centre en jugar, en lugar de en descifrar la interfaz de usuario.

We make the cursor visible again as soon as the game becomes inactive, which happens in `_ship_hit()` . Attention to details like this makes your game more professional looking and allows the player to focus on playing, rather than figuring out the user interface.

Pruébalo tú mismo (Try It Yourself)

14-1. Pulsa P para jugar: Como *Alien Invasion* utiliza la entrada del teclado para controlar la nave, sería útil iniciar el juego pulsando una tecla. Añade código que permita al jugador pulsar P para empezar. Podría ser útil mover parte del código de `_check_play_button()` a un método de `_start_game()` que pueda ser llamado desde `_check_play_button()` and `_check_keydown_events()` .

14-1. Press P to Play: Because *Alien Invasion* uses keyboard input to control the ship, it would be useful to start the game with a keypress. Add code that lets the player press P to start. It might help to move some code from `_check_play_button()` to a `_start_game()` method that can be called from `_check_play_button()` and `_check_keydown_events()` .

14-2. Práctica de puntería: Crea un rectángulo en el borde derecho de la pantalla que se mueva hacia arriba y hacia abajo a un ritmo constante. Luego, en el lado izquierdo de la pantalla, crea una nave que el jugador pueda mover arriba y abajo mientras dispara balas al objetivo rectangular. Añade un botón Jugar que inicie el juego, y cuando el jugador falle tres veces en el blanco, termina el juego y haz que vuelva a aparecer el botón Jugar. Deja que el jugador reinicie el juego con este botón Jugar.

14-2. Target Practice: Create a rectangle at the right edge of the screen that moves up and down at a steady rate. Then on the left side of the screen, create a ship that the player can move up and down while firing bullets at the rectangular target. Add a Play button that starts the game, and when the player misses the target three times, end the game and make the Play button reappear. Let the player restart the game with this Play button.

Subir de nivel (Leveling Up)

En nuestro juego actual, una vez que el jugador derriba toda la flota alienígena, alcanza un nuevo nivel, pero la dificultad del juego no cambia. Vamos a animar un poco las cosas y hacer que el juego sea más desafiante aumentando la velocidad del juego cada vez que el

jugador despeje la pantalla.

In our current game, once a player shoots down the entire alien fleet, the player reaches a new level, but the game difficulty doesn't change. Let's liven things up a bit and make the game more challenging by increasing the game's speed each time a player clears the screen.

Modificar los ajustes de velocidad (*Modifying the Speed Settings*)

Primero reorganizaremos la clase `Settings` para agrupar los ajustes del juego en estáticos y dinámicos. También nos aseguraremos de que los ajustes que cambien durante el juego se restablezcan cuando empecemos una nueva partida. Este es el método `__init__()` para *settings.py* :

We'll first reorganize the `Settings` class to group the game settings into static and dynamic ones. We'll also make sure any settings that change during the game reset when we start a new game. Here's the `__init__()` method for *settings.py* :

ajustes.py

settings.py

```
def __init__(self):
    """Initialize the game's static settings."""
    # Screen settings
    self.screen_width = 1200
    self.screen_height = 800
    self.bg_color = (230, 230, 230)

    # Ship settings
    self.ship_limit = 3

    # Bullet settings
    self.bullet_width = 3
    self.bullet_height = 15
    self.bullet_color = 60, 60, 60
    self.bullets_allowed = 3
```



```
# Alien settings
self.fleet_drop_speed = 10

# How quickly the game speeds up
❶ self.speedup_scale = 1.1

❷ self.initialize_dynamic_settings()
```

Seguimos inicializando ajustes que permanecen constantes en el método `__init__()` .
Añadimos un ajuste `speedup_scale` ❶ para controlar la velocidad a la que se acelera el juego: un valor de 2 duplicará la velocidad del juego cada vez que el jugador alcance un nuevo nivel; un valor de 1 mantendrá la velocidad constante. Un valor como 1.1 debería aumentar la velocidad lo suficiente como para que el juego sea desafiante pero no imposible. Por último, llamamos al método `initialize_dynamic_settings()` para inicializar los valores de los atributos que deben cambiar a lo largo del juego ❷ .

We continue to initialize settings that stay constant in the `__init__()` method. We add a `speedup_scale` setting ❶ to control how quickly the game speeds up: a value of 2 will double the game speed every time the player reaches a new level; a value of 1 will keep the speed constant. A value like 1.1 should increase the speed enough to make the game challenging but not impossible. Finally, we call the `initialize_dynamic_settings()` method to initialize the values for attributes that need to change throughout the game ❷ .

Este es el código de `initialize_dynamic_settings()` :

Here's the code for `initialize_dynamic_settings()` :

settings.py

settings.py

```
def initialize_dynamic_settings(self):
    """Initialize settings that change throughout the game."""
    self.ship_speed = 1.5
    self.bullet_speed = 2.5
```

```
self.alien_speed = 1.0
```

```
# fleet_direction of 1 represents right; -1 represents left.  
self.fleet_direction = 1
```

Este método establece los valores iniciales de las velocidades de la nave, la bala y el alienígena. Aumentaremos estas velocidades a medida que el jugador avance en el juego y las restableceremos cada vez que el jugador comience una nueva partida. Incluimos `fleet_direction` en este método para que los alienígenas se muevan siempre al principio de una nueva partida. No necesitamos aumentar el valor de `fleet_drop_speed`, porque cuando los alienígenas se muevan más rápido por la pantalla, también bajarán más rápido por la pantalla.

This method sets the initial values for the ship, bullet, and alien speeds. We'll increase these speeds as the player progresses in the game and reset them each time the player starts a new game. We include `fleet_direction` in this method so the aliens always move right at the beginning of a new game. We don't need to increase the value of `fleet_drop_speed`, because when the aliens move faster across the screen, they'll also come down the screen faster.

Para aumentar las velocidades de la nave, las balas y los alienígenas cada vez que el jugador alcance un nuevo nivel, escribiremos un nuevo método llamado `increase_speed()` :

To increase the speeds of the ship, bullets, and aliens each time the player reaches a new level, we'll write a new method called `increase_speed()` :

ajustes.py

settings.py

```
def increase_speed(self):  
    """Increase speed settings."""  
    self.ship_speed *= self.speedup_scale  
    self.bullet_speed *= self.speedup_scale  
    self.alien_speed *= self.speedup_scale
```

Para aumentar la velocidad de estos elementos del juego, multiplicamos cada ajuste de velocidad por el valor de `speedup_scale` .

To increase the speed of these game elements, we multiply each speed setting by the value of `speedup_scale` .

Aumentamos el ritmo del juego llamando a `increase_speed()` en `_check_bullet_alien_collisions()` cuando el último alienígena de una flota ha sido derribado:

We increase the game's tempo by calling `increase_speed()` in `_check_bullet_alien_collisions()` when the last alien in a fleet has been shot down:

alien_invasion.py

alien_invasion.py

```
def _check_bullet_alien_collisions(self):
    --snip--
    if not self.aliens:
        # Destroy existing bullets and create new fleet.
        self.bullets.empty()
        self._create_fleet()
        self.settings.increase_speed()
```

Cambiar los valores de los ajustes de velocidad `ship_speed` , `alien_speed` , y `bullet_speed` ; es suficiente para acelerar todo el juego!

Changing the values of the speed settings `ship_speed` , `alien_speed` , and `bullet_speed` is enough to speed up the entire game!

Restablecer la velocidad (*Resetting the Speed*)

Ahora tenemos que devolver los ajustes modificados a sus valores iniciales cada vez que el jugador comience una nueva partida; de lo contrario, cada nueva partida comenzaría con los ajustes de velocidad aumentados de la partida anterior:

Now we need to return any changed settings to their initial values each time the player starts a new game; otherwise, each new game would start with the increased speed settings of the previous game:

alien_invasion.py

alien_invasion.py

```
def _check_play_button(self, mouse_pos):
    """Start a new game when the player clicks Play."""
    button_clicked = self.play_button.rect.collidepoint(mouse_pos)
    if button_clicked and not self.game_active:
        # Reset the game settings.
        self.settings.initialize_dynamic_settings()
    --snip--
```

Jugar a *Alien Invasion* debería ser ahora más divertido y desafiante. Cada vez que despejes la pantalla, el juego debería acelerarse y volverse ligeramente más difícil. Si el juego se vuelve demasiado difícil demasiado rápido, disminuye el valor de `settings.speedup_scale` . O si el juego no es lo suficientemente difícil, aumenta ligeramente el valor. Encuentra un punto óptimo aumentando la dificultad en un tiempo razonable. Las primeras pantallas deben ser fáciles, las siguientes deben ser difíciles pero factibles, y las siguientes deben ser casi imposiblemente difíciles.

Playing *Alien Invasion* should be more fun and challenging now. Each time you clear the screen, the game should speed up and become slightly more difficult. If the game becomes too difficult too quickly, decrease the value of `settings.speedup_scale` . Or if the game isn't challenging enough, increase the value slightly. Find a sweet spot by ramping up the difficulty in a reasonable amount of time. The first couple of screens should be easy, the next few should be challenging but doable, and subsequent screens should be almost impossibly difficult.

Pruébalo tú mismo (*Try It Yourself*)

14-3. Práctica de objetivos desafiantes: Empieza con tu trabajo del Ejercicio 14-2 (página 283). Haz que el objetivo se mueva más rápido a medida que avanza el juego, y reinicia el objetivo a la velocidad original cuando el jugador haga clic en Jugar.

14-3. Challenging Target Practice: Start with your work from Exercise 14-2 (page 283). Make the target move faster as the game progresses, and restart the target at the original speed when the player clicks Play.

14-4. Niveles de Dificultad: Haz un conjunto de botones para *Alien Invasion* que permita al jugador seleccionar un nivel de dificultad inicial adecuado para el juego. Cada botón debe asignar los valores apropiados para los atributos en *Settings* necesarios para crear diferentes niveles de dificultad.

14-4. Difficulty Levels: Make a set of buttons for *Alien Invasion* that allows the player to select an appropriate starting difficulty level for the game. Each button should assign the appropriate values for the attributes in *Settings* needed to create different difficulty levels.

Puntuación (*Scoring*)

Implementemos un sistema de puntuación para seguir la puntuación del juego en tiempo real y mostrar la puntuación más alta, el nivel y el número de naves restantes.

Let's implement a scoring system to track the game's score in real time and display the high score, level, and number of ships remaining.

La puntuación es una estadística del juego, así que añadiremos un atributo `score` a `GameStats` :

The score is a game statistic, so we'll add a `score` attribute to `GameStats` :

game_stats.py

game_stats.py

```
class GameStats:
    --snip--
    def reset_stats(self):
        """Initialize statistics that can change during the game."""
        self.ships_left = self.ai_settings.ship_limit
        self.score = 0
```

Para restablecer la puntuación cada vez que comience una nueva partida, inicializaremos `score` en `reset_stats()` en lugar de `__init__()` .

To reset the score each time a new game starts, we initialize `score` in `reset_stats()` rather than `__init__()` .

Mostrar la puntuación (*Displaying the Score*)

Para mostrar la puntuación en la pantalla, primero creamos una nueva clase, `Scoreboard` . Por ahora, esta clase sólo mostrará la puntuación actual. Con el tiempo, también la utilizaremos para informar de la puntuación máxima, el nivel y el número de naves restantes. Aquí tienes la primera parte de la clase; guárdala como `scoreboard.py` :

To display the score on the screen, we first create a new class, `Scoreboard` . For now, this class will just display the current score. Eventually, we'll use it to report the high score, level, and number of ships remaining as well. Here's the first part of the class; save it as `scoreboard.py` :

marcador.py

scoreboard.py

```
import pygame.font

class Scoreboard:
    """A class to report scoring information."""

    ❶ def __init__(self, ai_game):
        """Initialize scorekeeping attributes."""
        self.screen = ai_game.screen
        self.screen_rect = self.screen.get_rect()
        self.settings = ai_game.settings
        self.stats = ai_game.stats

        # Font settings for scoring information.
        ❷ self.text_color = (30, 30, 30)
        ❸ self.font = pygame.font.SysFont(None, 48)

        # Prepare the initial score image.
        ❹ self.prep_score()
```

Como `Scoreboard` escribe texto en la pantalla, empezamos importando el módulo `pygame.font`. A continuación, damos a `__init__()` el parámetro `ai_game` para que pueda acceder a los objetos `settings`, `screen` y `stats`, que necesitará para informar de los valores que estamos siguiendo ❶. A continuación, establecemos un color de texto ❷ e instanciamos un objeto de fuente ❸.

Because `Scoreboard` writes text to the screen, we begin by importing the `pygame.font` module. Next, we give `__init__()` the `ai_game` parameter so it can access the `settings`, `screen`, and `stats` objects, which it will need to report the values we're tracking ❶. Then we set a text color ❷ and instantiate a font object ❸.

Para convertir el texto a mostrar en una imagen, llamamos a `prep_score()` ❹, que definimos aquí:

To turn the text to be displayed into an image, we call `prep_score()` ❹, which we define here:

marcador.py

scoreboard.py

```
def prep_score(self):
    """Turn the score into a rendered image."""
    ❶ score_str = str(self.stats.score)
    ❷ self.score_image = self.font.render(score_str, True,
        self.text_color, self.settings.bg_color)

    # Display the score at the top right of the screen.
    ❸ self.score_rect = self.score_image.get_rect()
    ❹ self.score_rect.right = self.screen_rect.right - 20
    ❺ self.score_rect.top = 20
```

En `prep_score()` , convertimos el valor numérico `stats.score` en una cadena ❶ y luego pasamos esta cadena a `render()` , que crea la imagen ❷ . Para mostrar claramente la puntuación en pantalla, pasamos el color de fondo de la pantalla y el color del texto a `render()` .

In `prep_score()` , we turn the numerical value `stats.score` into a string ❶ and then pass this string to `render()` , which creates the image ❷ . To display the score clearly onscreen, we pass the screen's background color and the text color to `render()` .

Colocaremos la puntuación en la esquina superior derecha de la pantalla y haremos que se expanda hacia la izquierda a medida que aumente la puntuación y crezca la anchura del número. Para asegurarnos de que la puntuación siempre se alinea con el lado derecho de la pantalla, creamos un `rect` llamado `score_rect` ❸ y establecemos su borde derecho a 20 píxeles del borde derecho de la pantalla ❹ . A continuación, colocamos el borde superior a 20 píxeles de la parte superior de la pantalla ❺ .

We'll position the score in the upper-right corner of the screen and have it expand to the left as the score increases and the width of the number grows. To make sure the score always lines up with the right side of the screen, we create a `rect` called `score_rect` ❸ and set its right edge 20 pixels from the right edge of the screen ❹ . We then place the top edge 20 pixels down from the top of the screen ❺ .

A continuación, creamos un método `show_score()` para mostrar la imagen de puntuación renderizada:

Then we create a `show_score()` method to display the rendered score image:

marcador.py

scoreboard.py

```
def show_score(self):  
    """Draw score to the screen."""  
    self.screen.blit(self.score_image, self.score_rect)
```

Este método dibuja la imagen del marcador en la pantalla en el lugar que especifique `score_rect`.

This method draws the score image onscreen at the location `score_rect` specifies.

Crear un marcador (*Making a Scoreboard*)

Para mostrar el marcador, crearemos una instancia de `Scoreboard` en `AlienInvasion`. En primer lugar, vamos a actualizar las sentencias `import`:

To display the score, we'll create a `Scoreboard` instance in `AlienInvasion`. First, let's update the `import` statements:

alien_invasion.py

alien_invasion.py

```
--snip--
from game_stats import GameStats
from scoreboard import Scoreboard
--snip--
```

A continuación, creamos una instancia de Scoreboard en `__init__()` :

Next, we make an instance of Scoreboard in `__init__()` :

invasión_extraterrestre.py

alien_invasion.py

```
def __init__(self):
    --snip--
    pygame.display.set_caption("Alien Invasion")

    # Create an instance to store game statistics,
    # and create a scoreboard.
    self.stats = GameStats(self)
    self.sb = Scoreboard(self)
    --snip--
```

Luego dibujamos el marcador en pantalla en `_update_screen()` :

Then we draw the scoreboard onscreen in `_update_screen()` :

invasión_extraterrestre.py

alien_invasion.py

```
def _update_screen(self):
    --snip--
```

```
self.aliens.draw(self.screen)

# Draw the score information.
self.sb.show_score()

# Draw the play button if the game is inactive.
--snip--
```

Llamamos a `show_score()` justo antes de dibujar el botón Reproducir.

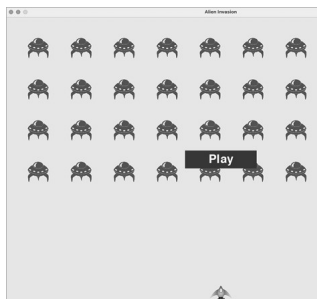
We call `show_score()` just before we draw the Play button.

Cuando ejecutes ahora *Alien Invasion*, debería aparecer un 0 en la parte superior derecha de la pantalla. (En este momento, sólo queremos asegurarnos de que la puntuación aparece en el lugar correcto antes de seguir desarrollando el sistema de puntuación). [La Figura 14-2](#) muestra la puntuación tal y como aparece antes de que comience el juego.

When you run *Alien Invasion* now, a 0 should appear at the top right of the screen. (At this point, we just want to make sure the score appears in the right place before developing the scoring system further.) [Figure 14-2](#) shows the score as it appears before the game starts.

A continuación, ¡asignaremos valores de puntos a cada alienígena!

Next, we'll assign point values to each alien!



DESCRIPCIÓN DE LA IMAGEN (En español):

La imagen muestra un juego estilo arcade con el título "Invasión Alienígena" en la parte superior. En el centro, hay numerosos iconos de naves alienígenas organizadas en filas. En el centro de la pantalla, se encuentra un botón con la palabra "Jugar".

En la esquina superior derecha, aparece un marcador con el número "0". En la parte inferior, hay un avatar o nave del jugador lista para comenzar el juego.

Figura 14-2 : La puntuación aparece en la esquina superior derecha de la pantalla.

Figure 14-2 : The score appears at the top-right corner of the screen.

Actualizar la puntuación a medida que se derriba a los alienígenas (*Updating the Score as Aliens Are Shot Down*)

Para escribir una puntuación en vivo en la pantalla, actualizamos el valor de `stats.score` cada vez que se derriba a un alienígena, y luego llamamos a `prep_score()` para actualizar la imagen de la puntuación. Pero antes, determinemos cuántos puntos obtiene un jugador cada vez que derriba a un alienígena:

To write a live score onscreen, we update the value of `stats.score` whenever an alien is hit, and then call `prep_score()` to update the score image. But first, let's determine how many points a player gets each time they shoot down an alien:

ajustes.py

settings.py

```
def initialize_dynamic_settings(self):
    --snip--

    # Scoring settings
    self.alien_points = 50
```

Aumentaremos el valor en puntos de cada alienígena a medida que avance la partida. Para asegurarnos de que este valor en puntos se reinicia cada vez que comienza una nueva partida, fijamos el valor en `initialize_dynamic_settings()` .

We'll increase each alien's point value as the game progresses. To make sure this point value is reset each time a new game starts, we set the value in `initialize_dynamic_settings()` .

Actualicemos la puntuación en `_check_bullet_alien_collisions()` cada vez que se derribe a un alienígena:

Let's update the score in `_check_bullet_alien_collisions()` each time an alien is shot down:

alien_invasion.py

alien_invasion.py

```
def _check_bullet_alien_collisions(self):
    """Respond to bullet-alien collisions."""
    # Remove any bullets and aliens that have collided.
    collisions = pygame.sprite.groupcollide(
        self.bullets, self.aliens, True, True)

    if collisions:
        self.stats.score += self.settings.alien_points
        self.sb.prep_score()
    --snip--
```

Cuando una bala alcanza a un alienígena, Pygame devuelve un diccionario `collisions` . Comprobamos si el diccionario existe, y si es así, el valor del alienígena se añade a la puntuación. A continuación, llamamos a `prep_score()` para crear una nueva imagen para la puntuación actualizada.

When a bullet hits an alien, Pygame returns a `collisions` dictionary. We check whether the dictionary exists, and if it does, the alien's value is added to the score. We then call `prep_score()` to create a new image for the updated score.

Ahora, cuando juegues a *Alien Invasion* , ¡podrás acumular puntos!

Now when you play *Alien Invasion* , you should be able to rack up points!

Restablecer la puntuación (*Resetting the Score*)

En este momento, sólo estamos preparando una nueva puntuación *after* se ha golpeado a un alienígena, lo que funciona durante la mayor parte del juego. Pero cuando empecemos una nueva partida, seguiremos viendo nuestra puntuación de la partida anterior hasta que sea alcanzado el primer alienígena.

Right now, we're only prepping a new score *after* an alien has been hit, which works for most of the game. But when we start a new game, we'll still see our score from the old game until the first alien is hit.

Podemos solucionarlo preparando la puntuación al iniciar una nueva partida:

We can fix this by prepping the score when starting a new game:

alien_invasion.py

alien_invasion.py

```
def _check_play_button(self, mouse_pos):
    --snip--
    if button_clicked and not self.game_active:
        --snip--
        # Reset the game statistics.
        self.stats.reset_stats()
        self.sb.prep_score()
        --snip--
```

Llamamos a `prep_score()` después de reiniciar las estadísticas del juego al iniciar una nueva partida. Esto prepara el marcador con una puntuación de 0.

We call `prep_score()` after resetting the game stats when starting a new game. This preps the scoreboard with a score of 0.

Asegurarse de anotar todos los golpes (*Making Sure to Score All Hits*)

Tal y como está escrito actualmente, nuestro código podría no puntuar a algunos alienígenas. Por ejemplo, si dos balas chocan con alienígenas durante la misma pasada por el bucle o si hacemos que una bala extraancha golpee a varios alienígenas, el jugador sólo recibirá puntos por golpear a uno de los alienígenas. Para solucionar esto, vamos a refinar la forma en que se detectan las colisiones bala-extraterrestre.

As currently written, our code could miss scoring for some aliens. For example, if two bullets collide with aliens during the same pass through the loop or if we make an extra-wide bullet to hit multiple aliens, the player will only receive points for hitting one of the aliens. To fix this, let's refine the way that bullet-alien collisions are detected.

En `_check_bullet_alien_collisions()`, cualquier bala que colisione con un alienígena se convierte en una clave del diccionario `collisions`. El valor asociado a cada bala es una lista de los alienígenas con los que ha colisionado. Recorremos en bucle los valores del diccionario `collisions` para asegurarnos de que otorgamos puntos por cada alienígena alcanzado:

In `_check_bullet_alien_collisions()`, any bullet that collides with an alien becomes a key in the `collisions` dictionary. The value associated with each bullet is a list of aliens it has collided with. We loop through the values in the `collisions` dictionary to make sure we award points for each alien hit:

alien_invasion.py

alien_invasion.py

```
def _check_bullet_alien_collisions(self):
    --snip--
    if collisions:
        for aliens in collisions.values():
            self.stats.score += self.settings.alien_points * len(aliens)
        self.sb.prep_score()
    --snip--
```

Si se ha definido el diccionario `collisions`, recorreremos todos los valores del diccionario. Recuerda que cada valor es una lista de alienígenas alcanzados por una sola bala. Multiplicamos el valor de cada alienígena por el número de alienígenas de cada lista y añadimos esta cantidad a la puntuación actual. Para probarlo, cambia la anchura de una bala a 300 píxeles y comprueba que recibes puntos por cada alienígena al que golpeas con tus balas extra anchas; luego devuelve la anchura de la bala a su valor normal.

If the `collisions` dictionary has been defined, we loop through all values in the dictionary. Remember that each value is a list of aliens hit by a single bullet. We multiply the value of each alien by the number of aliens in each list and add this amount to the current score. To test this, change the width of a bullet to 300 pixels and verify that you receive points for each alien you hit with your extra-wide bullets; then return the bullet width to its normal value.

Aumentar los valores de puntos *(Increasing Point Values)*

Dado que el juego se vuelve más difícil cada vez que el jugador alcanza un nuevo nivel, los alienígenas de los niveles posteriores deberían valer más puntos. Para implementar esta funcionalidad, añadiremos código para aumentar el valor en puntos cuando aumente la velocidad del juego:

Because the game gets more difficult each time a player reaches a new level, aliens in later levels should be worth more points. To implement this functionality, we'll add code to increase the point value when the game's speed increases:

settings.py

settings.py

```

class Settings:
    """A class to store all settings for Alien Invasion."""

    def __init__(self):
        --snip--
        # How quickly the game speeds up
        self.speedup_scale = 1.1
        # How quickly the alien point values increase
        ❶ self.score_scale = 1.5

        self.initialize_dynamic_settings()

    def initialize_dynamic_settings(self):
        --snip--

    def increase_speed(self):
        """Increase speed settings and alien point values."""
        self.ship_speed *= self.speedup_scale
        self.bullet_speed *= self.speedup_scale
        self.alien_speed *= self.speedup_scale

        ❷ self.alien_points = int(self.alien_points * self.score_scale)

```

Definimos una velocidad a la que aumentan los puntos, que llamamos `score_scale` ❶. Un pequeño aumento de la velocidad (1.1) hace que el juego sea más difícil rápidamente. Pero para ver una diferencia más notable en la puntuación, necesitamos cambiar el valor de los puntos alienígenas en una cantidad mayor (1.5). Ahora, cuando aumentamos la velocidad del juego, también aumentamos el valor en puntos de cada golpe ❷. Utilizamos la función `int()` para aumentar el valor en puntos en enteros.

We define a rate at which points increase, which we call `score_scale` ❶. A small increase in speed (1.1) makes the game more challenging quickly. But to see a more notable difference in scoring, we need to change the alien point value by a larger amount (1.5). Now when we increase the game's speed, we also increase the point value of each hit ❷. We use the `int()` function to increase the point value by whole integers.

Para ver el valor de cada alienígena, añade una llamada `print()` al método `increase_speed()` en `Settings` :

To see the value of each alien, add a `print()` call to the `increase_speed()` method in `Settings` :

ajustes.py

settings.py

```
def increase_speed(self):
    --snip--
    self.alien_points = int(self.alien_points * self.score_scale)
    print(self.alien_points)
```

El nuevo valor en puntos debería aparecer en el terminal cada vez que alcances un nuevo nivel.

The new point value should appear in the terminal every time you reach a new level.

Nota (Note)

Asegúrate de eliminar la llamada a `print()` después de comprobar que el valor en puntos aumenta, o podría afectar al rendimiento de tu juego y distraer al jugador.

Be sure to remove the `print()` call after verifying that the point value is increasing, or it might affect your game's performance and distract the player.

Redondear la puntuación (Rounding the Score)

La mayoría de los juegos de disparos de estilo arcade informan de las puntuaciones como múltiplos de 10, así que vamos a seguir esa pauta con nuestras puntuaciones. Además, formateemos la puntuación para incluir separadores de comas en los números grandes. Haremos este cambio en `Scoreboard` :

Most arcade-style shooting games report scores as multiples of 10, so let's follow that lead with our scores. Also, let's format the score to include comma separators in large numbers. We'll make this change in `Scoreboard` :

marcador.py

scoreboard.py

```
def prep_score(self):
    """Turn the score into a rendered image."""
    rounded_score = round(self.stats.score, -1)
    score_str = f"{rounded_score:,"}
    self.score_image = self.font.render(score_str, True,
        self.text_color, self.settings.bg_color)
    --snip--
```

La función `round()` normalmente redondea un flotante a un número determinado de decimales dado como segundo argumento. Sin embargo, cuando pasas un número negativo como segundo argumento, `round()` redondeará el valor al 10, 100, 1.000, etc., más próximos. Este código le dice a Python que redondee el valor de `stats.score` al 10 más próximo y lo asigne a `rounded_score` .

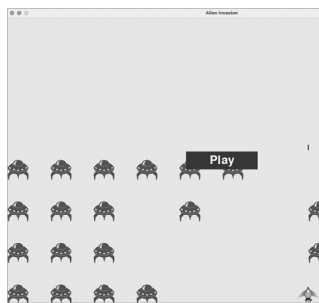
The `round()` function normally rounds a float to a set number of decimal places given as the second argument. However, when you pass a negative number as the second argument, `round()` will round the value to the nearest 10, 100, 1,000, and so on. This code tells Python to round the value of `stats.score` to the nearest 10 and assign it to `rounded_score` .

A continuación, utilizamos un especificador de formato en la cadena `f` para la puntuación. Un *format specifier* es una secuencia especial de caracteres que modifica la forma en que se presenta el valor de una variable. En este caso, la secuencia `;`, indica a Python que inserte comas en los lugares adecuados del valor numérico que se proporcione. Esto da como resultado cadenas como `1,000,000` en lugar de `1000000` .

We then use a format specifier in the f-string for the score. A *format specifier* is a special sequence of characters that modifies the way a variable's value is presented. In this case the sequence `;` tells Python to insert commas at appropriate places in the numerical value that's provided. This results in strings like `1,000,000` instead of `1000000`.

Ahora, cuando ejecutes el juego, deberías ver una puntuación redondeada y con un formato limpio, incluso cuando acumules muchos puntos, como se muestra en la [Figura 14-3](#).

Now when you run the game, you should see a neatly formatted, rounded score even when you rack up lots of points, as shown in [Figure 14-3](#).



DESCRIPCIÓN DE LA
IMAGEN (En español):

Invasión Alienígena
3,589,080 Jugar

Figura 14-3 : Puntuación redondeada con separadores de comas

Figure 14-3 : A rounded score with comma separators

Puntuaciones altas (High Scores)

Todos los jugadores quieren superar la puntuación máxima de un juego, así que vamos a registrar e informar de las puntuaciones máximas para dar a los jugadores algo por lo que esforzarse. Almacenaremos las puntuaciones altas en `GameStats` :

Every player wants to beat a game's high score, so let's track and report high scores to give players something to work toward. We'll store high scores in `GameStats` :

game_stats.py

game_stats.py

```
def __init__(self, ai_game):
    --snip--
    # High score should never be reset.
    self.high_score = 0
```

Como la puntuación máxima no debe restablecerse nunca, inicializamos `high_score` en `__init__()` en lugar de en `reset_stats()` .

Because the high score should never be reset, we initialize `high_score` in `__init__()` rather than in `reset_stats()` .

A continuación, modificaremos `Scoreboard` para que muestre la puntuación máxima. Empecemos con el método `__init__()` :

Next, we'll modify `Scoreboard` to display the high score. Let's start with the `__init__()` method:

marcador.py

scoreboard.py

```
def __init__(self, ai_game):
    --snip--
    # Prepare the initial score images.
    self.prep_score()
    ❶ self.prep_high_score()
```

La puntuación más alta se mostrará por separado de la puntuación, así que necesitamos un nuevo método, `prep_high_score()` , para preparar la imagen de la puntuación más alta ❶ .

The high score will be displayed separately from the score, so we need a new method, `prep_high_score()` , to prepare the high-score image ❶ .

Este es el método `prep_high_score()` :

Here's the `prep_high_score()` method:

marcador.py

scoreboard.py

```
def prep_high_score(self):
    """Turn the high score into a rendered image."""
    ❶ high_score = round(self.stats.high_score, -1)
    high_score_str = f'{high_score:,}'
    ❷ self.high_score_image = self.font.render(high_score_str, True,
        self.text_color, self.settings.bg_color)

    # Center the high score at the top of the screen.
    self.high_score_rect = self.high_score_image.get_rect()
    ❸ self.high_score_rect.centerx = self.screen_rect.centerx
    ❹ self.high_score_rect.top = self.score_rect.top
```

Redondeamos la puntuación más alta a la decena más próxima y le damos formato con comas ❶ . A continuación, generamos una imagen a partir de la puntuación máxima ❷ , centramos la puntuación máxima `rect` horizontalmente ❸ , y establecemos su atributo `top` para que coincida con la parte superior de la imagen de la puntuación ❹ .

We round the high score to the nearest 10 and format it with commas ❶ . We then generate an image from the high score ❷ , center the high score `rect` horizontally ❸ , and set its `top` attribute to match the top of the score image ❹ .

El método `show_score()` dibuja ahora la puntuación actual en la parte superior derecha y la puntuación alta en la parte superior central de la pantalla:

The `show_score()` method now draws the current score at the top right and the high score at the top center of the screen:

marcador.py

scoreboard.py

```
def show_score(self):
    """Draw score to the screen."""
    self.screen.blit(self.score_image, self.score_rect)
    self.screen.blit(self.high_score_image, self.high_score_rect)
```

Para comprobar las puntuaciones altas, escribiremos un nuevo método, `check_high_score()` , en `Scoreboard` :

To check for high scores, we'll write a new method, `check_high_score()` , in `Scoreboard` :

marcador.py

scoreboard.py

```
def check_high_score(self):
    """Check to see if there's a new high score."""
    if self.stats.score > self.stats.high_score:
        self.stats.high_score = self.stats.score
        self.prep_high_score()
```

El método `check_high_score()` comprueba la puntuación actual comparándola con la puntuación alta. Si la puntuación actual es mayor, actualizamos el valor de `high_score` y llamamos a `prep_high_score()` para actualizar la imagen de la puntuación más alta.

The method `check_high_score()` checks the current score against the high score. If the current score is greater, we update the value of `high_score` and call `prep_high_score()` to update the high score's image.

Tenemos que llamar a `check_high_score()` cada vez que se golpea a un alienígena después de actualizar la puntuación en `_check_bullet_alien_collisions()` :

We need to call `check_high_score()` each time an alien is hit after updating the score in `_check_bullet_alien_collisions()` :

alien_invasion.py

alien_invasion.py

```
def _check_bullet_alien_collisions(self):
    --snip--
    if collisions:
        for aliens in collisions.values():
            self.stats.score += self.settings.alien_points * len(aliens)
        self.sb.prep_score()
        self.sb.check_high_score()
    --snip--
```

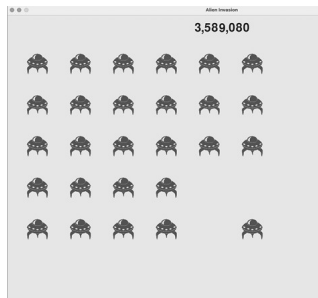
Llamamos a `check_high_score()` cuando está presente el diccionario `collisions` , y lo hacemos después de actualizar la puntuación de todos los alienígenas que han sido golpeados.

We call `check_high_score()` when the `collisions` dictionary is present, and we do so after updating the score for all the aliens that have been hit.

La primera vez que juegues a *Alien Invasion* , tu puntuación será la puntuación máxima, por lo que se mostrará como la puntuación actual y la puntuación máxima. Pero cuando empieces una segunda partida, tu puntuación alta debería aparecer en el centro y tu puntuación actual debería aparecer a la derecha, como se muestra en la [Figura 14-4](#) .

The first time you play *Alien Invasion* , your score will be the high score, so it will be displayed as the current score and the high score. But when you start a second game, your

high score should appear in the middle and your current score should appear at the right, as shown in [Figure 14-4](#).



DESCRIPCIÓN DE LA
IMAGEN (En español):

Alienígena Invasión
3,589,080 5,750

Figura 14-4 : La puntuación máxima aparece en la parte superior central de la pantalla.

Figure 14-4 : The high score is shown at the top center of the screen.

Visualización del nivel (*Displaying the Level*)

Para mostrar el nivel del jugador en el juego, primero necesitamos un atributo en `GameStats` que represente el nivel actual. Para restablecer el nivel al comienzo de cada nueva partida, inicialízalo en `reset_stats()` :

To display the player's level in the game, we first need an attribute in `GameStats` representing the current level. To reset the level at the start of each new game, initialize it in `reset_stats()` :

game_stats.py

game_stats.py

```
def reset_stats(self):
    """Initialize statistics that can change during the game."""
    self.ships_left = self.settings.ship_limit
    self.score = 0
    self.level = 1
```

Para que `Scoreboard` muestre el nivel actual, llamamos a un nuevo método, `prep_level()` , desde `__init__()` :

To have Scoreboard display the current level, we call a new method, `prep_level()`, from `__init__()` :

marcador.py

scoreboard.py

```
def __init__(self, ai_game):
    --snip--
    self.prep_high_score()
    self.prep_level()
```

Aquí tienes `prep_level()` :

Here's `prep_level()` :

marcador.py

scoreboard.py

```
def prep_level(self):
    """Turn the level into a rendered image."""
    level_str = str(self.stats.level)
    ❶ self.level_image = self.font.render(level_str, True,
        self.text_color, self.settings.bg_color)

    # Position the level below the score.
    self.level_rect = self.level_image.get_rect()
    ❷ self.level_rect.right = self.score_rect.right
    ❸ self.level_rect.top = self.score_rect.bottom + 10
```

El método `prep_level()` crea una imagen a partir del valor almacenado en `stats.level` ❶ y establece el atributo `right` de la imagen para que coincida con el atributo `right` de la puntuación ❷ . A continuación, establece el atributo `top` 10 píxeles por debajo de la parte inferior de la imagen del marcador para dejar espacio entre el marcador y el nivel ❸ .

The `prep_level()` method creates an image from the value stored in `stats.level` ❶ and sets the image's `right` attribute to match the score's `right` attribute ❷ . It then sets the `top` attribute 10 pixels beneath the bottom of the score image to leave space between the score and the level ❸ .

También tenemos que actualizar `show_score()` :

We also need to update `show_score()` :

marcador.py

scoreboard.py

```
def show_score(self):
    """Draw scores and level to the screen."""
    self.screen.blit(self.score_image, self.score_rect)
    self.screen.blit(self.high_score_image, self.high_score_rect)
    self.screen.blit(self.level_image, self.level_rect)
```

Esta nueva línea dibuja la imagen del nivel en la pantalla.

This new line draws the level image to the screen.

Incrementaremos `stats.level` y actualizaremos la imagen del nivel en `_check_bullet_alien_collisions()` :

We'll increment `stats.level` and update the level image in `_check_bullet_alien_collisions()` :

invasión_extraterrestre.py

alien_invasion.py

```
def _check_bullet_alien_collisions(self):
    --snip--
    if not self.aliens:
        # Destroy existing bullets and create new fleet.
        self.bullets.empty()
        self._create_fleet()
        self.settings.increase_speed()

        # Increase level.
        self.stats.level += 1
        self.sb.prep_level()
```

Si se destruye una flota, incrementamos el valor de `stats.level` y llamamos a `prep_level()` para asegurarnos de que el nuevo nivel se muestra correctamente.

If a fleet is destroyed, we increment the value of `stats.level` and call `prep_level()` to make sure the new level displays correctly.

Para asegurarnos de que la imagen del nivel se actualiza correctamente al inicio de una nueva partida, también llamamos a `prep_level()` cuando el jugador pulsa el botón Jugar:

To ensure the level image updates properly at the start of a new game, we also call `prep_level()` when the player clicks the Play button:

alien_invasion.py

alien_invasion.py

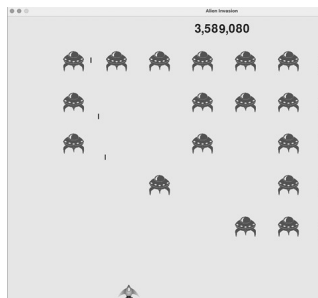
```
def _check_play_button(self, mouse_pos):
    --snip--
    if button_clicked and not self.game_active:
        --snip--
        self.sb.prep_score()
        self.sb.prep_level()
        --snip--
```

Llamamos a `prep_level()` justo después de llamar a `prep_score()` .

We call `prep_level()` right after calling `prep_score()` .

Ahora verás cuántos niveles has completado, como se muestra en la [Figura 14-5](#) .

Now you'll see how many levels you've completed, as shown in [Figure 14-5](#) .



DESCRIPCIÓN DE LA
IMAGEN (En español):

Invasión alienígena
3,589,080 51,650 8

Figura 14-5 : El nivel actual aparece justo debajo de la puntuación actual.

Figure 14-5 : The current level appears just below the current score.

Nota (Note)

En algunos juegos clásicos, las puntuaciones tienen etiquetas, como Puntuación, Puntuación máxima y Nivel. Hemos omitido estas etiquetas porque el significado de cada número queda claro una vez que has jugado al juego. Para incluir estas etiquetas, añádelas a las cadenas de puntuación justo antes de las llamadas a `font.render()` en `Scoreboard` .

In some classic games, the scores have labels, such as Score, High Score, and Level. We've omitted these labels because the meaning of each number becomes clear once you've played the game. To include these labels, add them to the score strings just before the calls to `font.render()` in `Scoreboard` .

Mostrar el número de naves (*Displaying the Number of Ships*)

Por último, vamos a mostrar el número de naves que le quedan al jugador, pero esta vez, utilizaremos un gráfico. Para ello, dibujaremos naves en la esquina superior izquierda de la pantalla para representar cuántas naves quedan, tal y como hacen muchos juegos arcade clásicos.

Finally, let's display the number of ships the player has left, but this time, let's use a graphic. To do so, we'll draw ships in the upper-left corner of the screen to represent how many ships are left, just as many classic arcade games do.

Primero, tenemos que hacer que `Ship` herede de `Sprite` para poder crear un grupo de naves:

First, we need to make `Ship` inherit from `Sprite` so we can create a group of ships:

nave.py

ship.py

```
import pygame
from pygame.sprite import Sprite
```

```
❶ class Ship(Sprite):
    """A class to manage the ship."""

    def __init__(self, ai_game):
        """Initialize the ship and set its starting position."""
```

```
❷    super().__init__()  
    --snip--
```

Aquí importamos `Sprite` , nos aseguramos de que `Ship` hereda de `Sprite` ❶ , y llamamos a `super()` al principio de `__init__()` ❷ .

Here we import `Sprite` , make sure `Ship` inherits from `Sprite` ❶ , and call `super()` at the beginning of `__init__()` ❷ .

A continuación, tenemos que modificar `Scoreboard` para crear un grupo de naves que podamos mostrar. Aquí están las sentencias `import` para `Scoreboard` :

Next, we need to modify `Scoreboard` to create a group of ships we can display. Here are the `import` statements for `Scoreboard` :

marcador.py

scoreboard.py

```
import pygame.font  
from pygame.sprite import Group  
  
from ship import Ship
```

Como estamos haciendo un grupo de barcos, importamos las clases `Group` y `Ship` .

Because we're making a group of ships, we import the `Group` and `Ship` classes.

Aquí tienes `__init__()` :

Here's `__init__()` :

marcador.py

scoreboard.py

```
def __init__(self, ai_game):
    """Initialize scorekeeping attributes."""
    self.ai_game = ai_game
    self.screen = ai_game.screen
    --snip--
    self.prep_level()
    self.prep_ships()
```

Asignamos la instancia del juego a un atributo, porque la necesitaremos para crear algunas naves. Llamamos a `prep_ships()` después de la llamada a `prep_level()` .

We assign the game instance to an attribute, because we'll need it to create some ships. We call `prep_ships()` after the call to `prep_level()` .

Aquí tienes `prep_ships()` :

Here's `prep_ships()` :

marcador.py

scoreboard.py

```
def prep_ships(self):
    """Show how many ships are left."""
    ❶ self.ships = Group()
    ❷ for ship_number in range(self.stats.ships_left):
        ship = Ship(self.ai_game)
```


- ③ `ship.rect.x = 10 + ship_number * ship.rect.width`
- ④ `ship.rect.y = 10`
- ⑤ `self.ships.add(ship)`

El método `prep_ships()` crea un grupo vacío, `self.ships`, para contener las instancias de las naves ①. Para llenar este grupo, se ejecuta un bucle una vez por cada nave que el jugador haya abandonado ②. Dentro del bucle, creamos una nueva nave y establecemos el valor de `x`-coordenada de cada nave para que las naves aparezcan una al lado de la otra con un margen de 10 píxeles a la izquierda del grupo de naves ③. Fijamos el valor de `y`-coordenada 10 píxeles hacia abajo desde la parte superior de la pantalla para que las naves aparezcan en la esquina superior izquierda de la pantalla ④. A continuación, añadimos cada nueva nave al grupo `ships` ⑤.

The `prep_ships()` method creates an empty group, `self.ships`, to hold the ship instances ①. To fill this group, a loop runs once for every ship the player has left ②. Inside the loop, we create a new ship and set each ship's `x`-coordinate value so the ships appear next to each other with a 10-pixel margin on the left side of the group of ships ③. We set the `y`-coordinate value 10 pixels down from the top of the screen so the ships appear in the upper-left corner of the screen ④. Then we add each new ship to the group `ships` ⑤.

Ahora tenemos que dibujar las naves en la pantalla:

Now we need to draw the ships to the screen:

marcador.py

scoreboard.py

```
def show_score(self):
    """Draw scores, level, and ships to the screen."""
    self.screen.blit(self.score_image, self.score_rect)
    self.screen.blit(self.high_score_image, self.high_score_rect)
    self.screen.blit(self.level_image, self.level_rect)
    self.ships.draw(self.screen)
```

Para mostrar las naves en la pantalla, llamamos a `draw()` en el grupo, y Pygame dibuja cada nave.

To display the ships on the screen, we call `draw()` on the group, and Pygame draws each ship.

Para mostrar al jugador cuántas naves tiene para empezar, llamamos a `prep_ships()` cuando comienza una nueva partida. Esto lo hacemos en `_check_play_button()` en `AlienInvasion` :

To show the player how many ships they have to start with, we call `prep_ships()` when a new game starts. We do this in `_check_play_button()` in `AlienInvasion` :

alien_invasion.py

alien_invasion.py

```
def _check_play_button(self, mouse_pos):
    --snip--
    if button_clicked and not self.game_active:
        --snip--
        self.sb.prep_level()
        self.sb.prep_ships()
        --snip--
```

También llamamos a `prep_ships()` cuando una nave es alcanzada, para actualizar la visualización de las imágenes de las naves cuando el jugador pierde una nave:

We also call `prep_ships()` when a ship is hit, to update the display of ship images when the player loses a ship:

alien_invasion.py

alien_invasion.py

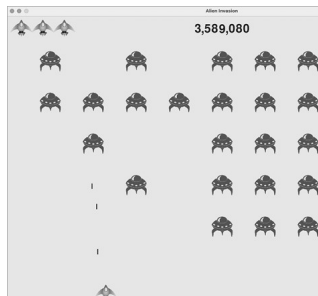
```
def _ship_hit(self):
    """Respond to ship being hit by alien."""
    if self.stats.ships_left > 0:
        # Decrement ships_left, and update scoreboard.
        self.stats.ships_left -= 1
        self.sb.prep_ships()
    --snip--
```

Llamamos a `prep_ships()` después de disminuir el valor de `ships_left`, para que se muestre el número correcto de naves restantes cada vez que se destruya una nave.

We call `prep_ships()` after decreasing the value of `ships_left`, so the correct number of remaining ships displays each time a ship is destroyed.

La [Figura 14-6](#) muestra el sistema de puntuación completo, con las naves restantes mostradas en la parte superior izquierda de la pantalla.

[Figure 14-6](#) shows the complete scoring system, with the remaining ships displayed at the top left of the screen.



DESCRIPCIÓN DE LA
IMAGEN (En español):

La imagen muestra una pantalla de un videojuego titulado "Invasión Alienígena". En la parte superior izquierda aparecen cinco iconos de naves pequeñas, mientras que a la derecha se ve el número "3,589,080" en una tipografía grande, posiblemente

indicando una puntuación o marcador. A la derecha, en la parte superior, está el número "302,100" seguido por el número "12" justo debajo. En el centro hay una serie de figuras que representan extraterrestres distribuidos en filas uniformes. En la parte inferior de la pantalla aparece una nave más grande acompañada de líneas verticales que podrían representar disparos o balas hacia los extraterrestres.

Figura 14-6 : El sistema de puntuación completo de Alien Invasion

Figure 14-6 : The complete scoring system for Alien Invasion

Pruébalo tú mismo (Try It Yourself)

14-5. Puntuación Máxima de Todos los Tiempos: La puntuación máxima se reinicia cada vez que un jugador cierra y reinicia *Alien Invasion* . Arréglalo escribiendo la puntuación máxima en un archivo antes de llamar a `sys.exit()` y leyendo la puntuación máxima al inicializar su valor en `GameStats` .

14-5. All-Time High Score: The high score is reset every time a player closes and restarts *Alien Invasion* . Fix this by writing the high score to a file before calling `sys.exit()` and reading in the high score when initializing its value in `GameStats` .

14-6. Refactorización: Busca métodos que estén haciendo más de una tarea, y refactorízalos para organizar tu código y hacerlo eficiente. Por ejemplo, mueve parte del código en `_check_bullet_alien_collisions()` , que inicia un nuevo nivel cuando la flota de alienígenas ha sido destruida, a una función llamada `start_new_level()` . Además, traslada las cuatro llamadas a métodos independientes del método `__init__()` de `Scoreboard` a un

método llamado `prep_images()` para acortar `__init__()` . El método `prep_images()` también podría ayudar a simplificar `_check_play_button()` o `start_game()` si ya has refactorizado `_check_play_button()` .

14-6. Refactoring: Look for methods that are doing more than one task, and refactor them to organize your code and make it efficient. For example, move some of the code in `_check_bullet_alien_collisions()` , which starts a new level when the fleet of aliens has been destroyed, to a function called `start_new_level()` . Also, move the four separate method calls in the `__init__()` method in `Scoreboard` to a method called `prep_images()` to shorten `__init__()` . The `prep_images()` method could also help simplify `_check_play_button()` or `start_game()` if you've already refactored `_check_play_button()` .

nota (note)

Antes de intentar refactorizar el proyecto, consulta el Apéndice D para aprender a restaurar el proyecto a un estado de funcionamiento si introduces errores mientras refactorizas.

Before attempting to refactor the project, see Appendix D to learn how to restore the project to a working state if you introduce bugs while refactoring.

14-7. Expandir el juego: Piensa en una forma de ampliar *Alien Invasion* . Por ejemplo, puedes programar a los alienígenas para que disparen balas a tu nave. También puedes añadir escudos para que tu nave se esconda detrás, que puedan ser destruidos por las balas de cualquier lado. O puedes utilizar algo como el módulo `pygame.mixer` para añadir efectos de sonido, como explosiones y sonidos de disparos.

14-7. Expanding the Game: Think of a way to expand *Alien Invasion* . For example, you could program the aliens to shoot bullets down at your ship. You can also add shields for your ship to hide behind, which can be destroyed by bullets from either side. Or you can use something like the `pygame.mixer` module to add sound effects, such as explosions and shooting sounds.

14-8. Tirador lateral, versión final: Continúa desarrollando *Sideways Shooter* , utilizando todo lo que hemos hecho en este proyecto. Añade un botón Jugar, haz que el juego se acelere en los puntos adecuados y desarrolla un sistema de puntuación. Asegúrate de refactorizar tu código a medida que trabajas, y busca oportunidades para personalizar el juego más allá de lo que se ha mostrado en este capítulo.

14-8. Sideways Shooter, Final Version: Continue developing *Sideways Shooter* , using everything we've done in this project. Add a Play button, make the game speed up at appropriate points, and develop a scoring system. Be sure to refactor your code as you work, and look for opportunities to customize the game beyond what has been shown in this chapter.

Resumen (Summary)

En este capítulo has aprendido a implementar un botón Jugar para iniciar una nueva partida. También has aprendido a detectar eventos del ratón y a ocultar el cursor en los juegos activos. Puedes utilizar lo que has aprendido para crear otros botones, como un botón Ayuda para mostrar instrucciones sobre cómo jugar a tus juegos. También aprendiste a modificar la velocidad de un juego a medida que avanza, a implementar un sistema de puntuación progresiva y a mostrar información de forma textual y no textual.

In this chapter, you learned how to implement a Play button to start a new game. You also learned how to detect mouse events and hide the cursor in active games. You can use what you've learned to create other buttons, like a Help button to display instructions on how to play your games. You also learned how to modify the speed of a game as it progresses, implement a progressive scoring system, and display information in textual and nontextual ways.

Capítulo 15: Generación de datos (Chapter 15: Generating Data)

15

Generar datos (15 Generating Data)



DESCRIPCIÓN DE LA
IMAGEN (En español):

La imagen muestra una ilustración de una serpiente enrollada alrededor de un motor a reacción. La serpiente tiene una expresión determinada, y el motor es claramente visible en la parte delantera. Los trazos son en blanco y negro, y el diseño es dinámico y estilizado, sugiriendo movimiento y velocidad.

Data visualization es el uso de representaciones visuales para explorar y presentar patrones en conjuntos de datos. Está estrechamente relacionado con *data analysis* , que utiliza código para explorar los patrones y conexiones de un conjunto de datos. Un conjunto de datos puede ser una pequeña lista de números que quepa en una sola línea de código, o pueden ser terabytes de datos que incluyan muchos tipos diferentes de información.

Data visualization is the use of visual representations to explore and present patterns in datasets. It's closely associated with *data analysis* , which uses code to explore the patterns and connections in a dataset. A dataset can be a small list of numbers that fits in a single line of code, or it can be terabytes of data that include many different kinds of information.

Crear visualizaciones de datos eficaces es algo más que hacer que la información tenga un aspecto agradable. Cuando la representación de un conjunto de datos es sencilla y visualmente atractiva, su significado resulta claro para los espectadores. La gente verá patrones y significados en tus conjuntos de datos que no sabían que existían.

Creating effective data visualizations is about more than just making information look nice. When a representation of a dataset is simple and visually appealing, its meaning becomes clear to viewers. People will see patterns and significance in your datasets that they never knew existed.

Afortunadamente, no necesitas un superordenador para visualizar datos complejos. Python es tan eficiente que, con un simple ordenador portátil, puedes explorar rápidamente conjuntos de datos que contengan millones de puntos de datos individuales. Estos puntos de datos no tienen por qué ser números; con las nociones básicas que aprendiste en la primera parte de este libro, también puedes analizar datos no numéricos.

Fortunately, you don't need a supercomputer to visualize complex data. Python is so efficient that with just a laptop, you can quickly explore datasets containing millions of individual data points. These data points don't have to be numbers; with the basics you learned in the first part of this book, you can analyze non-numerical data as well.

La gente utiliza Python para trabajos intensivos en datos en genética, investigación climática, análisis político y económico, y mucho más. Los científicos de datos han escrito una impresionante serie de herramientas de visualización y análisis en Python, muchas de las cuales también están a tu disposición. Una de las herramientas más populares es Matplotlib, una biblioteca de trazado matemático. En este capítulo, utilizaremos Matplotlib para hacer trazados sencillos, como gráficos de líneas y de dispersión. Luego crearemos un conjunto de datos más interesante basado en el concepto de paseo aleatorio: una visualización generada a partir de una serie de decisiones aleatorias.

People use Python for data-intensive work in genetics, climate research, political and economic analysis, and much more. Data scientists have written an impressive array of visualization and analysis tools in Python, many of which are available to you as well. One of the most popular tools is Matplotlib, a mathematical plotting library. In this chapter, we'll use Matplotlib to make simple plots, such as line graphs and scatter plots. Then we'll create a more interesting dataset based on the concept of a random walk—a visualization generated from a series of random decisions.

También utilizaremos un paquete llamado Plotly, que crea visualizaciones que funcionan bien en dispositivos digitales, para analizar los resultados de tirar dados. Plotly genera visualizaciones que cambian automáticamente de tamaño para adaptarse a diversos dispositivos de visualización. Estas visualizaciones también pueden incluir una serie de características interactivas, como resaltar aspectos concretos del conjunto de datos cuando

los usuarios pasan el ratón por encima de distintas partes de la visualización. Aprender a utilizar Matplotlib y Plotly te ayudará a empezar a visualizar los tipos de datos que más te interesan.

We'll also use a package called Plotly, which creates visualizations that work well on digital devices, to analyze the results of rolling dice. Plotly generates visualizations that automatically resize to fit a variety of display devices. These visualizations can also include a number of interactive features, such as emphasizing particular aspects of the dataset when users hover over different parts of the visualization. Learning to use Matplotlib and Plotly will help you get started visualizing the kinds of data you're most interested in.

Instalar Matplotlib (*Installing Matplotlib*)

Para utilizar Matplotlib en tu conjunto inicial de visualizaciones, tendrás que instalarlo mediante pip, igual que hicimos con pytest en el Capítulo 11 (consulta " Instalar pytest con pip " en la página 210).

To use Matplotlib for your initial set of visualizations, you'll need to install it using pip, just like we did with pytest in Chapter 11 (see " Installing pytest with pip " on page 210).

Para instalar Matplotlib, introduce el siguiente comando en una terminal:

To install Matplotlib, enter the following command at a terminal prompt:

```
$ python -m pip install --user matplotlib
```

Si utilizas un comando distinto de `python` para ejecutar programas o iniciar una sesión de terminal, como `python3` , tu comando tendrá este aspecto:

If you use a command other than `python` to run programs or start a terminal session, such as `python3` , your command will look like this:

```
$ python3 -m pip install --user matplotlib
```

Para ver los tipos de visualizaciones que puedes hacer con Matplotlib, visita la página principal de Matplotlib en <https://matplotlib.org> y haz clic en **Tipos de gráficos** . Cuando hagas clic en una visualización de la galería, verás el código utilizado para generar el gráfico.

To see the kinds of visualizations you can make with Matplotlib, visit the Matplotlib home page at <https://matplotlib.org> and click **Plot types** . When you click a visualization in the gallery, you'll see the code used to generate the plot.

Trazado de un gráfico lineal simple (*Plotting a Simple Line Graph*)

Vamos a trazar un gráfico lineal sencillo utilizando Matplotlib y luego lo personalizaremos para crear una visualización de datos más informativa. Utilizaremos la secuencia de números cuadrados 1, 4, 9, 16 y 25 como datos para el gráfico.

Let's plot a simple line graph using Matplotlib and then customize it to create a more informative data visualization. We'll use the square number sequence 1, 4, 9, 16, and 25 as the data for the graph.

Para hacer un gráfico lineal simple, especifica los números con los que quieres trabajar y deja que Matplotlib haga el resto:

To make a simple line graph, specify the numbers you want to work with and let Matplotlib do the rest:

mpl_cuadrados.py

mpl_squares.py

```
import matplotlib.pyplot as plt
```

```
squares = [1, 4, 9, 16, 25]
```

```
❶ fig, ax = plt.subplots()
```

```
ax.plot(squares)
```

```
plt.show()
```

Primero importamos el módulo `pyplot` utilizando el alias `plt` para no tener que escribir `pyplot` repetidamente. (Verás esta convención a menudo en los ejemplos en línea, así que la utilizaremos aquí.) El módulo `pyplot` contiene una serie de funciones que ayudan a generar gráficos y diagramas.

We first import the `pyplot` module using the alias `plt` so we don't have to type `pyplot` repeatedly. (You'll see this convention often in online examples, so we'll use it here.) The `pyplot` module contains a number of functions that help generate charts and plots.

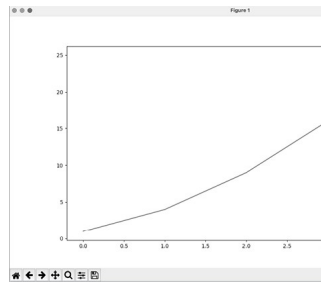
Creamos una lista llamada `squares` para guardar los datos que vamos a representar gráficamente. A continuación, seguimos otra convención común de Matplotlib llamando a la función `subplots()` ❶. Esta función puede generar uno o más gráficos en la misma figura. La variable `fig` representa la totalidad de *figure*, que es la colección de gráficos que se generan. La variable `ax` representa un único gráfico en la figura; ésta es la variable que utilizaremos la mayor parte del tiempo cuando definamos y personalizemos un único gráfico.

We create a list called `squares` to hold the data that we'll plot. Then we follow another common Matplotlib convention by calling the `subplots()` function ❶. This function can generate one or more plots in the same figure. The variable `fig` represents the entire *figure*, which is the collection of plots that are generated. The variable `ax` represents a single plot in the figure; this is the variable we'll use most of the time when defining and customizing a single plot.

A continuación, utilizamos el método `plot()`, que intenta trazar los datos que se le dan de una forma significativa. La función `plt.show()` abre el visor de Matplotlib y muestra el gráfico, como se muestra en [la Figura 15-1](#). El visor te permite hacer zoom y navegar por el gráfico, y puedes guardar las imágenes del gráfico que quieras haciendo clic en el icono del disco.

We then use the `plot()` method, which tries to plot the data it's given in a meaningful way. The function `plt.show()` opens Matplotlib's viewer and displays the plot, as shown in

Figure 15-1 . The viewer allows you to zoom and navigate the plot, and you can save any plot images you like by clicking the disk icon.



DESCRIPCIÓN DE LA IMAGEN (En español):

La imagen muestra un gráfico lineal simple en un entorno similar a una ventana de software para gráficos. En la parte superior, se presenta el título "Figura 1". El eje horizontal (eje x) tiene los números del 0.0 al 4.0 en incrementos de 0.5, mientras que el eje vertical (eje y) presenta los números del 0 al 25 en diversas escalas uniformes. La línea en el gráfico se eleva de manera constante de izquierda a derecha, comenzando en la coordenada (0, 0) y finalizando alrededor de (4, 25), lo que indica una tendencia de crecimiento positivo. Además, hay una barra de herramientas en la parte inferior con varios íconos pequeños que probablemente sirvan para manipular la visualización del gráfico, como acercar, alejar, mover y otras funciones relacionadas con la gráfica.

Figura 15-1 : Uno de los trazados más sencillos que puedes hacer en Matplotlib

Figure 15-1 : One of the simplest plots you can make in Matplotlib

Cambiar el tipo de etiqueta y el grosor de la línea *(Changing the Label Type and Line Thickness)*

Aunque el gráfico de [la Figura 15-1](#) muestra que los números están aumentando, el tipo de etiqueta es demasiado pequeño y la línea es un poco fina para leerla fácilmente. Afortunadamente, Matplotlib te permite ajustar todas las características de una visualización.

Although the plot in [Figure 15-1](#) shows that the numbers are increasing, the label type is too small and the line is a little thin to read easily. Fortunately, Matplotlib allows you to adjust every feature of a visualization.

Utilizaremos algunas de las personalizaciones disponibles para mejorar la legibilidad de este gráfico. Empecemos por añadir un título y etiquetar los ejes:

We'll use a few of the available customizations to improve this plot's readability. Let's start by adding a title and labeling the axes:

`mpl_cuadrados.py`

`mpl_squares.py`

```
import matplotlib.pyplot as plt

squares = [1, 4, 9, 16, 25]

fig, ax = plt.subplots()
❶ ax.plot(squares, linewidth=3)

# Set chart title and label axes.
❷ ax.set_title("Square Numbers", fontsize=24)
❸ ax.set_xlabel("Value", fontsize=14)
ax.set_ylabel("Square of Value", fontsize=14)

# Set size of tick labels.
❹ ax.tick_params(labelsize=14)
```

`plt.show()`

El parámetro `linewidth` controla el grosor de la línea que genera `plot()` ❶ . Una vez generado el gráfico, hay muchos métodos disponibles para modificarlo antes de presentarlo. El método `set_title()` establece un título general para el gráfico ❷ . Los parámetros `fontsize` , que aparecen repetidamente a lo largo del código, controlan el tamaño del texto en varios elementos del gráfico.

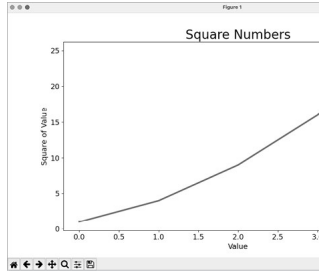
The `linewidth` parameter controls the thickness of the line that `plot()` generates ❶ . Once a plot has been generated, there are many methods available to modify the plot before it's presented. The `set_title()` method sets an overall title for the chart ❷ . The `fontsize` parameters, which appear repeatedly throughout the code, control the size of the text in various elements on the chart.

Los métodos `set_xlabel()` y `set_ylabel()` te permiten establecer un título para cada uno de los ejes ❸ , y el método `tick_params()` da estilo a las marcas de graduación ❹ . Aquí `tick_params()` establece el tamaño de fuente de las etiquetas de las marcas de graduación en 14 en ambos ejes.

The `set_xlabel()` and `set_ylabel()` methods allow you to set a title for each of the axes ❸ , and the method `tick_params()` styles the tick marks ❹ . Here `tick_params()` sets the font size of the tick mark labels to 14 on both axes.

Como puedes ver en [la Figura 15-2](#) , el gráfico resultante es mucho más fácil de leer. El tipo de etiqueta es más grande y el gráfico de líneas es más grueso. A menudo merece la pena experimentar con estos valores para ver qué funciona mejor en el gráfico resultante.

As you can see in [Figure 15-2](#) , the resulting chart is much easier to read. The label type is bigger, and the line graph is thicker. It's often worth experimenting with these values to see what works best in the resulting graph.



DESCRIPCIÓN DE LA IMAGEN (En español):

Figura 1 muestra un gráfico titulado "Números Cuadrados". En el eje horizontal se observa la etiqueta "Valor", mientras que en el eje vertical está la etiqueta "Cuadrado del Valor". El gráfico presenta una línea curva ascendente que ilustra la relación cuadrática entre los valores numéricos del eje horizontal y sus cuadrados en el eje vertical, con incrementos igualados a 0.5 en el eje de valor (de 0.0 a 4.0) y de 5 en el eje de cuadrado del valor (de 0 a 25).

Figura 15-2 : Ahora el gráfico es mucho más fácil de leer.

Figure 15-2 : The chart is much easier to read now.

Corregir el gráfico (Correcting the Plot)

Ahora que podemos leer mejor el gráfico, podemos ver que los datos no están trazados correctamente. Observa que, al final del gráfico, ¡el cuadrado de 4.0 aparece como 25! Vamos a corregirlo.

Now that we can read the chart better, we can see that the data is not plotted correctly. Notice at the end of the graph that the square of 4.0 is shown as 25! Let's fix that.

Cuando das a `plot()` una única secuencia de números, asume que el primer punto de datos corresponde a un valor x de 0, pero nuestro primer punto corresponde a un valor x de 1. Podemos anular el comportamiento por defecto dando a `plot()` tanto los valores de entrada como los de salida utilizados para calcular los cuadrados:

When you give `plot()` a single sequence of numbers, it assumes the first data point corresponds to an `x`-value of 0, but our first point corresponds to an `x`-value of 1. We can override the default behavior by giving `plot()` both the input and output values used to calculate the squares:

`mpl_cuadrados.py`

`mpl_squares.py`

```
import matplotlib.pyplot as plt

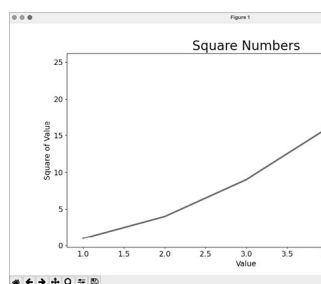
input_values = [1, 2, 3, 4, 5]
squares = [1, 4, 9, 16, 25]

fig, ax = plt.subplots()
ax.plot(input_values, squares, linewidth=3)

# Set chart title and label axes.
--snip--
```

Ahora `plot()` no tiene que hacer ninguna suposición sobre cómo se generaron los números de salida. El gráfico resultante, que se muestra en [la Figura 15-3](#), es correcto.

Now `plot()` doesn't have to make any assumptions about how the output numbers were generated. The resulting plot, shown in [Figure 15-3](#), is correct.



DESCRIPCIÓN DE LA
IMAGEN (En español):

Figura 1 Números
Cuadrados Cuadrado
del Valor Valor

Figura 15-3 : Ahora los datos se representan correctamente.

Figure 15-3 : The data is now plotted correctly.

Puedes especificar una serie de argumentos al llamar a `plot()` y utilizar una serie de métodos para personalizar tus gráficos después de generarlos. Seguiremos explorando estos métodos de personalización a medida que trabajemos con conjuntos de datos más interesantes a lo largo de este capítulo.

You can specify a number of arguments when calling `plot()` and use a number of methods to customize your plots after generating them. We'll continue to explore these approaches to customization as we work with more interesting datasets throughout this chapter.

Uso de estilos incorporados *(Using Built-in Styles)*

Matplotlib dispone de una serie de estilos predefinidos. Estos estilos contienen una variedad de ajustes predeterminados para colores de fondo, líneas de cuadrícula, anchos de línea, fuentes, tamaños de fuente y mucho más. Pueden hacer que tus visualizaciones sean atractivas sin requerir mucha personalización. Para ver la lista completa de estilos disponibles, ejecuta las siguientes líneas en una sesión de terminal:

Matplotlib has a number of predefined styles available. These styles contain a variety of default settings for background colors, gridlines, line widths, fonts, font sizes, and more. They can make your visualizations appealing without requiring much customization. To see the full list of available styles, run the following lines in a terminal session:

```
>>> import matplotlib.pyplot as plt
>>> plt.style.available
['Solarize_Light2', '_classic_test_patch', '_mpl-gallery',
--snip--
```

Para utilizar cualquiera de estos estilos, añade una línea de código antes de llamar a `subplots()` :

To use any of these styles, add one line of code before calling `subplots()` :

`mpl_cuadrados.py`

`mpl_squares.py`

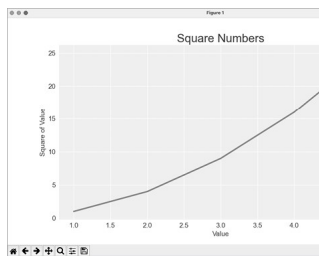
```
import matplotlib.pyplot as plt
```

```
input_values = [1, 2, 3, 4, 5]  
squares = [1, 4, 9, 16, 25]
```

```
plt.style.use('seaborn')  
fig, ax = plt.subplots()  
--snip--
```

Este código genera el gráfico que se muestra en [la Figura 15-4](#) . Hay una gran variedad de estilos disponibles; juega con ellos para encontrar alguno que te guste.

This code generates the plot shown in [Figure 15-4](#) . A wide variety of styles is available; play around with these styles to find some that you like.



DESCRIPCIÓN DE LA
IMAGEN (En español):

Figura 1 Números
Cuadrados Cuadrado
del Valor Valor

Figura 15-4 : Estilo seaborn incorporado

Figure 15-4 : The built-in seaborn style

Trazar y estilizar puntos individuales con `scatter()` (*Plotting and Styling Individual Points with `scatter()`*)

A veces resulta útil trazar y estilizar puntos individuales en función de determinadas características. Por ejemplo, puedes trazar valores pequeños en un color y valores más grandes en otro. También puedes trazar un gran conjunto de datos con un conjunto de opciones de estilo y, a continuación, resaltar puntos individuales volviendo a trazarlos con opciones diferentes.

Sometimes, it's useful to plot and style individual points based on certain characteristics. For example, you might plot small values in one color and larger values in a different color. You could also plot a large dataset with one set of styling options and then emphasize individual points by replotting them with different options.

Para trazar un único punto, pasa los valores x - y y - del punto a `scatter()` :

To plot a single point, pass the single x - and y -values of the point to `scatter()` :

scatter_squares.py

scatter_squares.py

```
import matplotlib.pyplot as plt
```

```
plt.style.use('seaborn')
fig, ax = plt.subplots()
ax.scatter(2, 4)
```

```
plt.show()
```

Estilicemos la salida para hacerla más interesante. Añadiremos un título, etiquetaremos los ejes y nos aseguraremos de que todo el texto sea lo suficientemente grande como para poder leerlo:

Let's style the output to make it more interesting. We'll add a title, label the axes, and make sure all the text is large enough to read:

```
import matplotlib.pyplot as plt
```

```
plt.style.use('seaborn')
fig, ax = plt.subplots()
❶ ax.scatter(2, 4, s=200)
```

```
# Set chart title and label axes.
```

```
ax.set_title("Square Numbers", fontsize=24)
```

```
ax.set_xlabel("Value", fontsize=14)
```

```
ax.set_ylabel("Square of Value", fontsize=14)
```

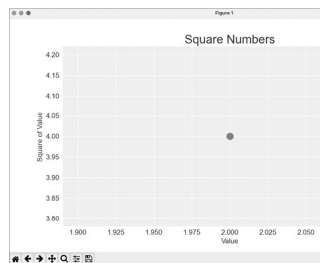
```
# Set size of tick labels.
```

```
ax.tick_params(labelsize=14)
```

```
plt.show()
```

Llamamos a `scatter()` y utilizamos el argumento `s` para establecer el tamaño de los puntos utilizados para dibujar el gráfico ❶. Cuando ejecute ahora `scatter_squares.py`, debería ver un único punto en el centro del gráfico, como se muestra en la [Figura 15-5](#).

We call `scatter()` and use the `s` argument to set the size of the dots used to draw the graph ❶. When you run `scatter_squares.py` now, you should see a single point in the middle of the chart, as shown in [Figure 15-5](#).



DESCRIPCIÓN DE LA
IMAGEN (En español):

En la imagen se muestra un gráfico con el título "Números Cuadrados". El eje horizontal está etiquetado como "Valor" y el eje vertical como "Cuadrado de Valor". El gráfico presenta un solo punto destacado en las coordenadas donde ambos valores son 2.000 y 4.000.

Figura 15-5 : Trazado de un único punto

Figure 15-5 : Plotting a single point

Trazar una serie de puntos con `scatter()` (*Plotting a Series of Points with `scatter()`*)

Para trazar una serie de puntos, podemos pasar a `scatter()` listas separadas de valores `x` - `y`, de la siguiente manera:

To plot a series of points, we can pass `scatter()` separate lists of `x` - and `y` -values, like this:

`scatter_cuadrados.py`

scatter_squares.py

```
import matplotlib.pyplot as plt

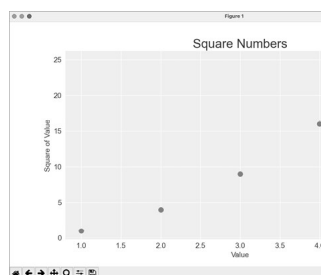
x_values = [1, 2, 3, 4, 5]
y_values = [1, 4, 9, 16, 25]

plt.style.use('seaborn')
fig, ax = plt.subplots()
ax.scatter(x_values, y_values, s=100)

# Set chart title and label axes.
--snip--
```

La lista `x_values` contiene los números que hay que elevar al cuadrado, y `y_values` contiene el cuadrado de cada número. Cuando estas listas se pasan a `scatter()`, Matplotlib lee un valor de cada lista al trazar cada punto. Los puntos que se trazan son (1, 1), (2, 4), (3, 9), (4, 16) y (5, 25); [la Figura 15-6](#) muestra el resultado.

The `x_values` list contains the numbers to be squared, and `y_values` contains the square of each number. When these lists are passed to `scatter()`, Matplotlib reads one value from each list as it plots each point. The points to be plotted are (1, 1), (2, 4), (3, 9), (4, 16), and (5, 25); [Figure 15-6](#) shows the result.



DESCRIPCIÓN DE LA IMAGEN (En español):

La imagen muestra un gráfico de dispersión titulado "Números Cuadrados". En el eje horizontal se etiqueta como "Valor", mientras que el eje vertical lleva la etiqueta "Cuadrado del Valor". Se observan puntos dispersos que reflejan la relación entre los valores del eje X y sus correspondientes cuadrados en el eje Y.

Figura 15-6 : Diagrama de dispersión con múltiples puntos

Figure 15-6 : A scatter plot with multiple points

Calcular datos automáticamente (Calculating Data Automatically)

Escribir listas a mano puede resultar ineficaz, sobre todo cuando tenemos muchos puntos. En lugar de escribir cada valor, utilicemos un bucle que haga los cálculos por nosotros.

Writing lists by hand can be inefficient, especially when we have many points. Rather than writing out each value, let's use a loop to do the calculations for us.

Así se vería esto con 1.000 puntos:

Here's how this would look with 1,000 points:

esparcir_cuadrados.py

scatter_squares.py

```
import matplotlib.pyplot as plt

❶ x_values = range(1, 1001)
y_values = [x**2 for x in x_values]

plt.style.use('seaborn')
fig, ax = plt.subplots()
❷ ax.scatter(x_values, y_values, s=10)

# Set chart title and label axes.
--snip--

# Set the range for each axis.
❸ ax.axis([0, 1100, 0, 1_100_000])

plt.show()
```

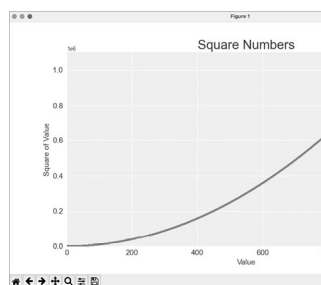
Empezamos con un rango de x -valores que contienen los números del 1 al 1.000 ❶ . A continuación, una comprensión de listas genera los y -valores recorriendo en bucle los x

-valores (`for x in x_values`), elevando al cuadrado cada número (`x**2`) y asignando los resultados a `y_values` . A continuación, pasamos las listas de entrada y salida a `scatter()` ❷ . Como se trata de un conjunto de datos grande, utilizamos un tamaño de punto más pequeño.

We start with a range of x -values containing the numbers 1 through 1,000 ❶ . Next, a list comprehension generates the y -values by looping through the x -values (`for x in x_values`), squaring each number (`x**2`), and assigning the results to `y_values` . We then pass the input and output lists to `scatter()` ❷ . Because this is a large dataset, we use a smaller point size.

Antes de mostrar el gráfico, utilizamos el método `axis()` para especificar el rango de cada eje ❸ . El método `axis()` requiere cuatro valores: los valores mínimo y máximo del eje x y del eje y . Aquí, ejecutamos el eje x de 0 a 1.100 y el eje y de 0 a 1.100.000. [La Figura 15-7](#) muestra el resultado.

Before showing the plot, we use the `axis()` method to specify the range of each axis ❸ . The `axis()` method requires four values: the minimum and maximum values for the x -axis and the y -axis. Here, we run the x -axis from 0 to 1,100 and the y -axis from 0 to 1,100,000. [Figure 15-7](#) shows the result.



DESCRIPCIÓN DE LA IMAGEN (En español):

La imagen muestra un gráfico titulado "Números Cuadrados". En el eje horizontal, se puede observar el término "Valor" y en el eje vertical se menciona "Cuadrado del Valor". El gráfico presenta una línea curva que representa el crecimiento del cuadrado de un número a medida que aumenta el valor. La escala del eje vertical va desde 0 hasta 1.0 x 10⁶, indicando el

valor cuadrado, mientras que la escala del eje horizontal va de 0 a 1000, mostrando el valor original. En la parte superior de la imagen, se encuentra un título "Figura 1".

Figura 15-7 : Python puede trazar 1.000 puntos tan fácilmente como traza 5 puntos.

Figure 15-7 : Python can plot 1,000 points as easily as it plots 5 points.

Personalizar las etiquetas de los puntos (Customizing Tick Labels)

Cuando los números de un eje son lo suficientemente grandes, Matplotlib utiliza por defecto la notación científica para las etiquetas de las marcas. Esto suele ser bueno, porque los números más grandes en notación simple ocupan mucho espacio innecesario en una visualización.

When the numbers on an axis get large enough, Matplotlib defaults to scientific notation for tick labels. This is usually a good thing, because larger numbers in plain notation take up a lot of unnecessary space on a visualization.

Casi todos los elementos de un gráfico son personalizables, así que puedes decirle a Matplotlib que siga utilizando la notación plana si lo prefieres:

Almost every element of a chart is customizable, so you can tell Matplotlib to keep using plain notation if you prefer:

```
--snip--  
# Set the range for each axis.  
ax.axis([0, 1100, 0, 1_100_000])  
ax.ticklabel_format(style='plain')
```

```
plt.show()
```

El método `ticklabel_format()` te permite anular el estilo predeterminado de la etiqueta de marca para cualquier gráfico.

The `ticklabel_format()` method allows you to override the default tick label style for any plot.

Definir colores personalizados *(Defining Custom Colors)*

Para cambiar el color de los puntos, pasa el argumento `color` a `scatter()` con el nombre de un color a utilizar entre comillas, como se muestra aquí:

To change the color of the points, pass the argument `color` to `scatter()` with the name of a color to use in quotation marks, as shown here:

```
ax.scatter(x_values, y_values, color='red', s=10)
```

También puedes definir colores personalizados utilizando el modelo de color RGB. Para definir un color, pasa al argumento `color` una tupla con tres valores flotantes (uno para el rojo, otro para el verde y otro para el azul, en ese orden), utilizando valores entre 0 y 1. Por ejemplo, la línea siguiente crea un gráfico con puntos de color verde claro:

You can also define custom colors using the RGB color model. To define a color, pass the `color` argument a tuple with three float values (one each for red, green, and blue, in that order), using values between 0 and 1. For example, the following line creates a plot with light-green dots:

```
ax.scatter(x_values, y_values, color=(0, 0.8, 0), s=10)
```

Los valores más cercanos a 0 producen colores más oscuros, y los valores más cercanos a 1 producen colores más claros.

Values closer to 0 produce darker colors, and values closer to 1 produce lighter colors.

Utilizar un mapa de colores *(Using a Colormap)*

Un *colormap* es una secuencia de colores en un gradiente que se mueve de un color inicial a uno final. En las visualizaciones, los mapas de colores se utilizan para resaltar patrones en los datos. Por ejemplo, puedes hacer que los valores bajos tengan un color claro y los valores altos un color más oscuro. El uso de un mapa de colores garantiza que todos los

puntos de la visualización varíen suavemente y con precisión a lo largo de una escala de colores bien diseñada.

A *colormap* is a sequence of colors in a gradient that moves from a starting to an ending color. In visualizations, colormaps are used to emphasize patterns in data. For example, you might make low values a light color and high values a darker color. Using a colormap ensures that all points in the visualization vary smoothly and accurately along a well-designed color scale.

El módulo `pyplot` incluye un conjunto de mapas de colores integrados. Para utilizar uno de estos mapas de colores, tienes que especificar cómo `pyplot` debe asignar un color a cada punto del conjunto de datos. A continuación se explica cómo asignar un color a cada punto, basándose en su valor `y` :

The `pyplot` module includes a set of built-in colormaps. To use one of these colormaps, you need to specify how `pyplot` should assign a color to each point in the dataset. Here's how to assign a color to each point, based on its `y` -value:

scatter_squares.py

scatter_squares.py

```
--snip--
plt.style.use('seaborn')
fig, ax = plt.subplots()
ax.scatter(x_values, y_values, c=y_values, cmap=plt.cm.Blues, s=10)

# Set chart title and label axes.
--snip--
```

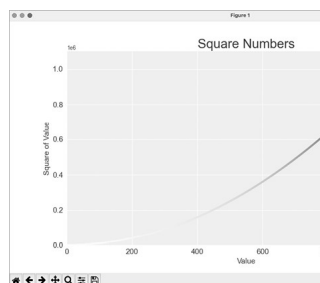
El argumento `c` es similar a `color` , pero se utiliza para asociar una secuencia de valores a una asignación de color. Pasamos la lista de `y` -valores a `c` , y luego le dice a `pyplot` qué mapa de color utilizar con el argumento `cmap` . Este código colorea de azul claro los puntos con valores `y` más bajos y de azul oscuro los puntos con valores `y` más altos. [La Figura 15-8](#) muestra el gráfico resultante.

The `c` argument is similar to `color` , but it's used to associate a sequence of values with a color mapping. We pass the list of γ -values to `c` , and then tell `pyplot` which colormap to use with the `cmap` argument. This code colors the points with lower γ -values light blue and the points with higher γ -values dark blue. [Figure 15-8](#) shows the resulting plot.

Nota (Note)

Puedes ver todos los mapas de colores disponibles en `pyplot` en <https://matplotlib.org> . Ve a Tutoriales, desplázate hasta Colores y haz clic en **Elegir mapas de colores en Matplotlib** .

You can see all the colormaps available in `pyplot` at <https://matplotlib.org> . Go to Tutorials, scroll down to Colors, and click **Choosing Colormaps in Matplotlib** .



DESCRIPCIÓN DE LA IMAGEN (En español):

La imagen presenta un gráfico titulado "Números Cuadrados".

En el eje horizontal está la etiqueta "Valor", mientras que en el eje vertical se lee "Cuadrado del Valor".

El gráfico muestra una curva ascendente en color negro que representa el crecimiento cuadrático, comenzando suavemente y volviéndose más empinada a medida que avanza hacia una cifra cercana a 1000 en el eje horizontal, y alcanzando alrededor

de 1.0e6 en el eje
vertical.

Figura 15-8 : Un gráfico utilizando el mapa de colores Blues

Figure 15-8 : A plot using the Blues colormap

Guardar tus trazados automáticamente (Saving Your Plots Automatically)

Si quieres guardar el gráfico en un archivo en lugar de mostrarlo en el visor de Matplotlib, puedes utilizar `plt.savefig()` en lugar de `plt.show()` :

If you want to save the plot to a file instead of showing it in the Matplotlib viewer, you can use `plt.savefig()` instead of `plt.show()` :

```
plt.savefig('squares_plot.png', bbox_inches='tight')
```

El primer argumento es un nombre de archivo para la imagen del gráfico, que se guardará en el mismo directorio que `scatter_squares.py` . El segundo argumento recorta los espacios en blanco sobrantes del gráfico. Si quieres el espacio en blanco extra alrededor del gráfico, puedes omitir este argumento. También puedes llamar a `savefig()` con un objeto `Path` , y escribir el archivo de salida en cualquier lugar que desees de tu sistema.

The first argument is a filename for the plot image, which will be saved in the same directory as `scatter_squares.py` . The second argument trims extra whitespace from the plot. If you want the extra whitespace around the plot, you can omit this argument. You can also call `savefig()` with a `Path` object, and write the output file anywhere you want on your system.

Pruébalo tú mismo (Try It Yourself)

15-1. Cubos: Un número elevado a la tercera potencia es un *cube* . Traza los cinco primeros números cúbicos, y luego traza los 5.000 primeros números cúbicos.

15-1. Cubes: A number raised to the third power is a *cube* . Plot the first five cubic numbers, and then plot the first 5,000 cubic numbers.

15-2. Cubos coloreados: Aplica un mapa de colores a tu gráfico de cubos.

15-2. Colored Cubes: Apply a colormap to your cubes plot.

Recorridos aleatorios (*Random Walks*)

En esta sección, utilizaremos Python para generar datos para un paseo aleatorio y luego utilizaremos Matplotlib para crear una representación visualmente atractiva de esos datos. Un *random walk* es un camino determinado por una serie de decisiones simples, cada una de las cuales se deja totalmente al azar. Podrías imaginar un paseo aleatorio como el camino que seguiría una hormiga confundida si diera cada paso en una dirección aleatoria.

In this section, we'll use Python to generate data for a random walk and then use Matplotlib to create a visually appealing representation of that data. A *random walk* is a path that's determined by a series of simple decisions, each of which is left entirely to chance. You might imagine a random walk as the path a confused ant would take if it took every step in a random direction.

Los paseos aleatorios tienen aplicaciones prácticas en la naturaleza, la física, la biología, la química y la economía. Por ejemplo, un grano de polen flotando en una gota de agua se mueve por la superficie del agua porque es empujado constantemente por las moléculas de agua. El movimiento molecular en una gota de agua es aleatorio, por lo que la trayectoria que traza un grano de polen sobre la superficie es un paseo aleatorio. El código que escribiremos a continuación modela muchas situaciones del mundo real.

Random walks have practical applications in nature, physics, biology, chemistry, and economics. For example, a pollen grain floating on a drop of water moves across the surface of the water because it's constantly pushed around by water molecules. Molecular motion in a water drop is random, so the path a pollen grain traces on the surface is a random walk. The code we'll write next models many real-world situations.

Creación de la clase *RandomWalk* (*Creating the RandomWalk Class*)

Para crear un paseo aleatorio, crearemos una clase `RandomWalk`, que tomará decisiones aleatorias sobre la dirección que debe tomar el paseo. La clase necesita tres atributos: una

variable para registrar el número de puntos del paseo, y dos listas para almacenar las coordenadas x - y y - de cada punto del paseo.

To create a random walk, we'll create a `RandomWalk` class, which will make random decisions about which direction the walk should take. The class needs three attributes: one variable to track the number of points in the walk, and two lists to store the x - and y -coordinates of each point in the walk.

Sólo necesitaremos dos métodos para la clase `RandomWalk` : el método `__init__()` y `fill_walk()` , que calculará los puntos del paseo. Empecemos por el método `__init__()` :

We'll only need two methods for the `RandomWalk` class: the `__init__()` method and `fill_walk()` , which will calculate the points in the walk. Let's start with the `__init__()` method:

paseo_aleatorio.py

random_walk.py

❶ from random import choice

class RandomWalk:

"""A class to generate random walks."""

❷ def __init__(self, num_points=5000):

"""Initialize attributes of a walk."""

self.num_points = num_points

All walks start at (0, 0).

❸ self.x_values = [0]

self.y_values = [0]

Para tomar decisiones aleatorias, almacenaremos los posibles movimientos en una lista y utilizaremos la función `choice()` (del módulo `random`) para decidir qué movimiento hacer cada vez que se dé un paso ❶ . Establecemos por defecto el número de puntos de un paseo en `5000` , que es lo suficientemente grande como para generar algunos patrones

interesantes, pero lo suficientemente pequeño como para generar paseos rápidamente ❷ . A continuación, creamos dos listas para guardar los valores x - y , y comenzamos cada recorrido en el punto (0, 0) ❸ .

To make random decisions, we'll store possible moves in a list and use the `choice()` function (from the `random` module) to decide which move to make each time a step is taken ❶ . We set the default number of points in a walk to 5000 , which is large enough to generate some interesting patterns but small enough to generate walks quickly ❷ . Then we make two lists to hold the x - and y -values, and we start each walk at the point (0, 0) ❸ .

Elección de direcciones (*Choosing Directions*)

Utilizaremos el método `fill_walk()` para determinar la secuencia completa de puntos del paseo. Añade este método a *random_walk.py* :

We'll use the `fill_walk()` method to determine the full sequence of points in the walk. Add this method to *random_walk.py* :

paseo_aleatorio.py

random_walk.py

```
def fill_walk(self):
    """Calculate all the points in the walk."""

    # Keep taking steps until the walk reaches the desired length.
    ❶ while len(self.x_values) < self.num_points:

        # Decide which direction to go, and how far to go.
        ❷ x_direction = choice([1, -1])
        x_distance = choice([0, 1, 2, 3, 4])
        ❸ x_step = x_direction * x_distance

        y_direction = choice([1, -1])
        y_distance = choice([0, 1, 2, 3, 4])
```

```

❹      y_step = y_direction * y_distance

      # Reject moves that go nowhere.
❺      if x_step == 0 and y_step == 0:
          continue

      # Calculate the new position.
❻      x = self.x_values[-1] + x_step
          y = self.y_values[-1] + y_step

      self.x_values.append(x)
      self.y_values.append(y)

```

Primero establecemos un bucle que se ejecuta hasta que el paseo se llena con el número correcto de puntos ❶ . La parte principal de `fill_walk()` le dice a Python cómo simular cuatro decisiones aleatorias: ¿Irá el paseo hacia la derecha o hacia la izquierda? ¿Qué distancia recorrerá en esa dirección? ¿Irá hacia arriba o hacia abajo? ¿Qué distancia recorrerá en esa dirección?

We first set up a loop that runs until the walk is filled with the correct number of points ❶ . The main part of `fill_walk()` tells Python how to simulate four random decisions: Will the walk go right or left? How far will it go in that direction? Will it go up or down? How far will it go in that direction?

Utilizamos `choice([1, -1])` para elegir un valor para `x_direction` , que devuelve 1 para el movimiento hacia la derecha o -1 para el movimiento hacia la izquierda ❷ . A continuación, `choice([0, 1, 2, 3, 4])` selecciona aleatoriamente una distancia para moverse en esa dirección. Asignamos este valor a `x_distance` . La inclusión de un 0 permite la posibilidad de pasos que tengan movimiento a lo largo de un solo eje.

We use `choice([1, -1])` to choose a value for `x_direction` , which returns either 1 for movement to the right or -1 for movement to the left ❷ . Next, `choice([0, 1, 2, 3, 4])` randomly selects a distance to move in that direction. We assign this value to `x_distance` . The inclusion of a 0 allows for the possibility of steps that have movement along only one axis.

Determinamos la longitud de cada paso en las direcciones x- y y- multiplicando la dirección del movimiento por la distancia elegida ❸ ❹ . Un resultado positivo para `x_step` significa moverse hacia la derecha, un resultado negativo significa moverse hacia la

izquierda, y 0 significa moverse verticalmente. Un resultado positivo para `y_step` significa moverse hacia arriba, negativo significa moverse hacia abajo, y 0 significa moverse horizontalmente. Si los valores tanto de `x_step` como de `y_step` son 0, el recorrido no va a ninguna parte; cuando esto ocurre, continuamos el bucle ⑤ .

We determine the length of each step in the *x*- and *y*- directions by multiplying the direction of movement by the distance chosen ③ ④ . A positive result for `x_step` means move to the right, a negative result means move to the left, and 0 means move vertically. A positive result for `y_step` means move up, negative means move down, and 0 means move horizontally. If the values of both `x_step` and `y_step` are 0, the walk doesn't go anywhere; when this happens, we continue the loop ⑤ .

Para obtener el siguiente *x* -valor del paseo, sumamos el valor de `x_step` al último valor almacenado en `x_values` ⑥ y hacemos lo mismo con los *y* -valores. Cuando tengamos las coordenadas del nuevo punto, las añadimos a `x_values` y `y_values` .

To get the next *x* -value for the walk, we add the value in `x_step` to the last value stored in `x_values` ⑥ and do the same for the *y* -values. When we have the new point's coordinates, we append them to `x_values` and `y_values` .

Trazar el recorrido aleatorio (*Plotting the Random Walk*)

Aquí tienes el código para trazar todos los puntos del paseo:

Here's the code to plot all the points in the walk:

rw_visual.py

rw_visual.py

```
import matplotlib.pyplot as plt
```

```
from random_walk import RandomWalk
```

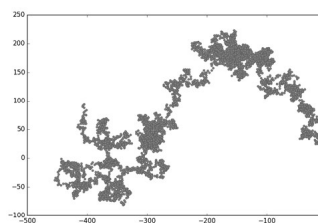
```
# Make a random walk.  
❶ rw = RandomWalk()  
rw.fill_walk()  
  
# Plot the points in the walk.  
plt.style.use('classic')  
fig, ax = plt.subplots()  
❷ ax.scatter(rw.x_values, rw.y_values, s=15)  
❸ ax.set_aspect('equal')  
plt.show()
```

Empezamos importando `pyplot` y `RandomWalk` . A continuación, creamos un paseo aleatorio y lo asignamos a `rw` ❶ , asegurándonos de llamar a `fill_walk()` . Para visualizar el paseo, introducimos los valores `x` - y `y` - del paseo en `scatter()` y elegimos un tamaño de punto adecuado ❷ . Por defecto, Matplotlib escala cada eje de forma independiente. Pero ese método estiraría la mayoría de los recorridos horizontal o verticalmente. Aquí utilizamos el método `set_aspect()` para especificar que ambos ejes deben tener el mismo espaciado entre marcas de graduación ❸ .

We begin by importing `pyplot` and `RandomWalk` . We then create a random walk and assign it to `rw` ❶ , making sure to call `fill_walk()` . To visualize the walk, we feed the walk's `x` - and `y` -values to `scatter()` and choose an appropriate dot size ❷ . By default, Matplotlib scales each axis independently. But that approach would stretch most walks out horizontally or vertically. Here we use the `set_aspect()` method to specify that both axes should have equal spacing between tick marks ❸ .

La [Figura 15-9](#) muestra el gráfico resultante con 5.000 puntos. Las imágenes de esta sección omiten el visor de Matplotlib, pero seguirás viéndolo cuando ejecutes `rw_visual.py` .

[Figure 15-9](#) shows the resulting plot with 5,000 points. The images in this section omit Matplotlib's viewer, but you'll continue to see it when you run `rw_visual.py` .



DESCRIPCIÓN DE LA
IMAGEN (En español):

La imagen muestra un gráfico en blanco y negro de un posible proceso de caminata aleatoria en dos dimensiones. La

trayectoria está representada por una línea irregular que se mueve de manera aparentemente aleatoria a través del espacio, formando una estructura compleja y ramificada. El eje horizontal, etiquetado como "x-axis", se extiende desde -500 hasta 100. El eje vertical, etiquetado como "y-axis", va desde 0 hasta 250. Ambos ejes están numerados en intervalos regulares. No hay texto adicional en la imagen.

Figura 15-9 : Un paseo aleatorio con 5.000 puntos

Figure 15-9 : A random walk with 5,000 points

Generar múltiples paseos aleatorios (*Generating Multiple Random Walks*)

Cada paseo aleatorio es diferente, y es divertido explorar los distintos patrones que se pueden generar. Una forma de utilizar el código anterior para hacer múltiples paseos sin tener que ejecutar el programa varias veces es envolverlo en un bucle `while`, como éste:

Every random walk is different, and it's fun to explore the various patterns that can be generated. One way to use the preceding code to make multiple walks without having to run the program several times is to wrap it in a `while` loop, like this:

rw_visual.py

rw_visual.py

```
import matplotlib.pyplot as plt
```

```
from random_walk import RandomWalk

# Keep making new walks, as long as the program is active.
while True:
    # Make a random walk.
    --snip--
    plt.show()

    keep_running = input("Make another walk? (y/n): ")
    if keep_running == 'n':
        break
```

Este código genera un paseo aleatorio, lo muestra en el visor de Matplotlib y hace una pausa con el visor abierto. Cuando cierres el visor, se te preguntará si quieres generar otro paseo. Si generas unos cuantos paseos, deberías ver algunos que se quedan cerca del punto de partida, otros que se alejan sobre todo en una dirección, otros que tienen tramos delgados que conectan grupos de puntos más grandes, y muchos otros tipos de paseos. Cuando quieras terminar el programa, pulsa N.

This code generates a random walk, displays it in Matplotlib's viewer, and pauses with the viewer open. When you close the viewer, you'll be asked whether you want to generate another walk. If you generate a few walks, you should see some that stay near the starting point, some that wander off mostly in one direction, some that have thin sections connecting larger groups of points, and many other kinds of walks. When you want to end the program, press N.

Estilizar el recorrido (*Styling the Walk*)

En esta sección, personalizaremos nuestros trazados para resaltar las características importantes de cada paseo y restar importancia a los elementos que distraen. Para ello, identificamos las características que queremos resaltar, como dónde empezó el paseo, dónde terminó y el camino recorrido. A continuación, identificamos las características a las que queremos restar importancia, como las marcas de graduación y las etiquetas. El resultado debe ser una representación visual sencilla que comunique claramente el camino recorrido en cada paseo aleatorio.

In this section, we'll customize our plots to emphasize the important characteristics of each walk and deemphasize distracting elements. To do so, we identify the characteristics we want to emphasize, such as where the walk began, where it ended, and the path taken. Next, we identify the characteristics to deemphasize, such as tick marks and labels. The

result should be a simple visual representation that clearly communicates the path taken in each random walk.

Colorear los puntos (Coloring the Points)

Utilizaremos un mapa de colores para mostrar el orden de los puntos en el paseo, y eliminaremos el contorno negro de cada punto para que el color de los puntos sea más claro. Para colorear los puntos según su posición en el recorrido, pasamos al argumento `c` una lista que contiene la posición de cada punto. Como los puntos se trazan en orden, esta lista sólo contiene los números de 0 a 4,999:

We'll use a colormap to show the order of the points in the walk, and remove the black outline from each dot so the color of the dots will be clearer. To color the points according to their position in the walk, we pass the `c` argument a list containing the position of each point. Because the points are plotted in order, this list just contains the numbers from 0 to 4,999:

rw_visual.py

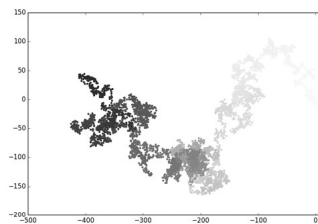
rw_visual.py

```
--snip--
while True:
    # Make a random walk.
    rw = RandomWalk()
    rw.fill_walk()

    # Plot the points in the walk.
    plt.style.use('classic')
    fig, ax = plt.subplots()
    ❶ point_numbers = range(rw.num_points)
    ax.scatter(rw.x_values, rw.y_values, c=point_numbers, cmap=plt.cm.Blues,
               edgecolors='none', s=15)
    ax.set_aspect('equal')
    plt.show()
--snip--
```

Utilizamos `range()` para generar una lista de números igual al número de puntos del recorrido ❶. Asignamos esta lista a `point_numbers`, que utilizaremos para establecer el color de cada punto del paseo. Pasamos `point_numbers` al argumento `c`, utilizamos el mapa de colores `Blues`, y luego pasamos `edgecolors='none'` para deshacernos del contorno negro alrededor de cada punto. El resultado es un gráfico que varía del azul claro al azul oscuro, mostrando exactamente cómo se mueve el paseo desde su punto inicial hasta su punto final. Esto se muestra en la [Figura 15-10](#).

We use `range()` to generate a list of numbers equal to the number of points in the walk ❶. We assign this list to `point_numbers`, which we'll use to set the color of each point in the walk. We pass `point_numbers` to the `c` argument, use the `Blues` colormap, and then pass `edgecolors='none'` to get rid of the black outline around each point. The result is a plot that varies from light to dark blue, showing exactly how the walk moves from its starting point to its ending point. This is shown in [Figure 15-10](#).



DESCRIPCIÓN DE LA
IMAGEN (En español):

La imagen muestra un gráfico de líneas de puntos en diferentes tonos de gris sobre un fondo cuadriculado. El eje horizontal va desde -500 hasta 100, mientras que el eje vertical abarca desde -200 hasta 150. Los puntos se extienden principalmente en una dirección diagonal ascendente de izquierda a derecha, comenzando con tonos oscuros que se aclaran progresivamente hacia el centro y el lado derecho de la imagen.

Figura 15-10 : Un paseo aleatorio coloreado con el mapa de colores `Blues`

Figure 15-10 : A random walk colored with the `Blues` colormap

Trazar los puntos inicial y final (Plotting the Starting and Ending Points)

Además de colorear los puntos para mostrar su posición a lo largo del recorrido, sería útil ver exactamente dónde empieza y acaba cada recorrido. Para ello, podemos trazar los puntos inicial y final individualmente, una vez trazada la serie principal. Haremos los puntos finales más grandes y los colorearemos de forma diferente para que destaquen:

In addition to coloring points to show their position along the walk, it would be useful to see exactly where each walk begins and ends. To do so, we can plot the first and last points individually after the main series has been plotted. We'll make the end points larger and color them differently to make them stand out:

rw_visual.py

rw_visual.py

```
--snip--
while True:
    --snip--
    ax.scatter(rw.x_values, rw.y_values, c=point_numbers, cmap=plt.cm.Blues,
               edgecolors='none', s=15)
    ax.set_aspect('equal')

    # Emphasize the first and last points.
    ax.scatter(0, 0, c='green', edgecolors='none', s=100)
    ax.scatter(rw.x_values[-1], rw.y_values[-1], c='red', edgecolors='none',
               s=100)

    plt.show()
--snip--
```

Para mostrar el punto inicial, trazamos el punto (0, 0) en verde y en un tamaño mayor (s=100) que el resto de los puntos. Para marcar el punto final, trazamos los últimos valores x - y y - también en rojo y con un tamaño de 100. Asegúrate de insertar este código justo antes de la llamada a plt.show() para que los puntos inicial y final se dibujen encima de todos los demás puntos.

To show the starting point, we plot the point (0, 0) in green and in a larger size (`s=100`) than the rest of the points. To mark the end point, we plot the last `x` - and `y` -values in red with a size of 100 as well. Make sure you insert this code just before the call to `plt.show()` so the starting and ending points are drawn on top of all the other points.

Cuando ejecutes este código, deberías ser capaz de distinguir exactamente dónde empieza y acaba cada paseo. Si estos puntos finales no destacan con suficiente claridad, ajusta su color y tamaño hasta que lo hagan.

When you run this code, you should be able to spot exactly where each walk begins and ends. If these end points don't stand out clearly enough, adjust their color and size until they do.

Limpiar los ejes (Cleaning Up the Axes)

Eliminemos los ejes de este gráfico para que no distraigan de la trayectoria de cada paseo. He aquí cómo ocultar los ejes:

Let's remove the axes in this plot so they don't distract from the path of each walk. Here's how to hide the axes:

rw_visual.py

rw_visual.py

```
--snip--
while True:
    --snip--
    ax.scatter(rw.x_values[-1], rw.y_values[-1], c='red', edgecolors='none',
               s=100)

    # Remove the axes.
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
```



```
plt.show()
--snip--
```

Para modificar los ejes, utilizamos los métodos `ax.get_xaxis()` y `ax.get_yaxis()` para obtener cada eje, y luego encadenamos el método `set_visible()` para hacer invisible cada eje. A medida que sigas trabajando con visualizaciones, verás con frecuencia este encadenamiento de métodos para personalizar distintos aspectos de una visualización.

To modify the axes, we use the `ax.get_xaxis()` and `ax.get_yaxis()` methods to get each axis, and then chain the `set_visible()` method to make each axis invisible. As you continue to work with visualizations, you'll frequently see this chaining of methods to customize different aspects of a visualization.

Ejecuta ahora `rw_visual.py` ; deberías ver una serie de gráficos sin ejes.

Run `rw_visual.py` now; you should see a series of plots with no axes.

Añadir puntos de trazado (Adding Plot Points)

Aumentemos el número de puntos, para tener más datos con los que trabajar. Para ello, aumentamos el valor de `num_points` cuando hagamos una instancia de `RandomWalk` y ajustamos el tamaño de cada punto al dibujar el gráfico:

Let's increase the number of points, to give us more data to work with. To do so, we increase the value of `num_points` when we make a `RandomWalk` instance and adjust the size of each dot when drawing the plot:

`rw_visual.py`

`rw_visual.py`

```
--snip--
while True:
    # Make a random walk.
```

```

rw = RandomWalk(50_000)
rw.fill_walk()

# Plot the points in the walk.
plt.style.use('classic')
fig, ax = plt.subplots()
point_numbers = range(rw.num_points)
ax.scatter(rw.x_values, rw.y_values, c=point_numbers, cmap=plt.cm.Blues,
          edgecolors='none', s=1)
--snip--

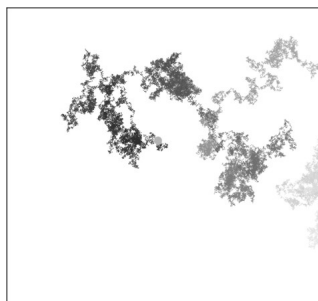
```

Este ejemplo crea un paseo aleatorio con 50.000 puntos y traza cada punto con el tamaño `s=1` . El paseo resultante es difuso y con forma de nube, como se muestra en la [Figura 15-11](#) . ¡Hemos creado una obra de arte a partir de un simple gráfico de dispersión!

This example creates a random walk with 50,000 points and plots each point at size `s=1` . The resulting walk is wispy and cloudlike, as shown in [Figure 15-11](#) . We've created a piece of art from a simple scatter plot!

Experimenta con este código para ver cuánto puedes aumentar el número de puntos de un recorrido antes de que tu sistema empiece a ralentizarse significativamente o el gráfico pierda su atractivo visual.

Experiment with this code to see how much you can increase the number of points in a walk before your system starts to slow down significantly or the plot loses its visual appeal.



DESCRIPCIÓN DE LA IMAGEN (En español):

La imagen muestra un diseño abstracto que parece ser un patrón fractal o un camino aleatorio. Se compone de una serie de formas intrincadas en diferentes tonos de gris, que se extienden desde un área más oscura en la parte izquierda hacia una más clara en la

derecha. También se observa un punto negro en la parte inferior derecha, que contrasta con el fondo blanco. La configuración parece tener un flujo u orientación hacia la derecha.

Figura 15-11 : Un paseo con 50.000 puntos

Figure 15-11 : A walk with 50,000 points

Alterar el tamaño para llenar la pantalla (Altering the Size to Fill the Screen)

Una visualización es mucho más eficaz a la hora de comunicar patrones en los datos si se ajusta bien a la pantalla. Para que la ventana de trazado se ajuste mejor a tu pantalla, puedes ajustar el tamaño de la salida de Matplotlib. Esto se hace en la llamada `subplots()` :

A visualization is much more effective at communicating patterns in data if it fits nicely on the screen. To make the plotting window better fit your screen, you can adjust the size of Matplotlib's output. This is done in the `subplots()` call:

```
fig, ax = plt.subplots(figsize=(15, 9))
```

Al crear un gráfico, puedes pasar a `subplots()` un argumento `figsize` , que establece el tamaño de la figura. El parámetro `figsize` toma una tupla que indica a Matplotlib las dimensiones de la ventana de trazado en pulgadas.

When creating a plot, you can pass `subplots()` a `figsize` argument, which sets the size of the figure. The `figsize` parameter takes a tuple that tells Matplotlib the dimensions of the plotting window in inches.

Matplotlib asume que la resolución de tu pantalla es de 100 píxeles por pulgada; si este código no te da un tamaño de trazado exacto, ajusta los números según sea necesario. O, si conoces la resolución de tu sistema, puedes pasar a `subplots()` la resolución utilizando el parámetro `dpi` :

Matplotlib assumes your screen resolution is 100 pixels per inch; if this code doesn't give you an accurate plot size, adjust the numbers as necessary. Or, if you know your system's resolution, you can pass `subplots()` the resolution using the `dpi` parameter:

```
fig, ax = plt.subplots(figsize=(10, 6), dpi=128)
```

Esto debería ayudarte a hacer un uso más eficiente del espacio disponible en tu pantalla.

This should help make the most efficient use of the space available on your screen.

Pruébalo tú mismo (*Try It Yourself*)

15-3. Movimiento molecular: Modifica *rw_visual.py* sustituyendo `ax.scatter()` por `ax.plot()` . Para simular la trayectoria de un grano de polen sobre la superficie de una gota de agua, pasa `rw.x_values` y `rw.y_values` , e incluye un argumento `linewidth` . Utiliza 5.000 puntos en lugar de 50.000 para que el gráfico no quede demasiado recargado.

15-3. Molecular Motion: Modify *rw_visual.py* by replacing `ax.scatter()` with `ax.plot()` . To simulate the path of a pollen grain on the surface of a drop of water, pass in the `rw.x_values` and `rw.y_values` , and include a `linewidth` argument. Use 5,000 instead of 50,000 points to keep the plot from being too busy.

15-4. Paseos aleatorios modificados: En la clase `RandomWalk` , `x_step` y `y_step` se generan a partir del mismo conjunto de condiciones. La dirección se elige aleatoriamente de la lista `[1, -1]` y la distancia de la lista `[0, 1, 2, 3, 4]` . Modifica los valores de estas listas para ver qué ocurre con la forma general de tus paseos. Prueba con una lista más larga de opciones para la distancia, como de 0 a 8, o elimina el -1 de la lista `x-` o `y-` dirección.

15-4. Modified Random Walks: In the `RandomWalk` class, `x_step` and `y_step` are generated from the same set of conditions. The direction is chosen randomly from the list `[1, -1]` and the distance from the list `[0, 1, 2, 3, 4]` . Modify the values in these lists to see what happens to the overall shape of your walks. Try a longer list of choices for the distance, such as 0 through 8, or remove the -1 from the `x-` or `y-` direction list.

15-5. Refactorización: El método `fill_walk()` es largo. Crea un nuevo método llamado `get_step()` para determinar la dirección y la distancia de cada paso, y luego calcula el paso. Deberías acabar con dos llamadas a `get_step()` en `fill_walk()` :

15-5. Refactoring: The `fill_walk()` method is lengthy. Create a new method called `get_step()` to determine the direction and distance for each step, and then calculate the step. You should end up with two calls to `get_step()` in `fill_walk()` :

```
x_step = self.get_step()
y_step = self.get_step()
```

Esta refactorización debería reducir el tamaño de `fill_walk()` y facilitar la lectura y comprensión del método.

This refactoring should reduce the size of `fill_walk()` and make the method easier to read and understand.

Tirar dados con Plotly (*Rolling Dice with Plotly*)

En esta sección, utilizaremos Plotly para producir visualizaciones interactivas. Plotly es especialmente útil cuando creas visualizaciones que se mostrarán en un navegador, porque las visualizaciones se escalarán automáticamente para ajustarse a la pantalla del espectador. Estas visualizaciones también son interactivas; cuando el usuario pasa el ratón por encima de determinados elementos de la pantalla, se resalta la información sobre esos elementos. Construiremos nuestra visualización inicial en sólo un par de líneas de código utilizando *Plotly Express*, un subconjunto de Plotly que se centra en generar gráficos con el menor código posible. Una vez que sepamos que nuestro trazado es correcto, personalizaremos la salida igual que hicimos con Matplotlib.

In this section, we'll use Plotly to produce interactive visualizations. Plotly is particularly useful when you're creating visualizations that will be displayed in a browser, because the visualizations will scale automatically to fit the viewer's screen. These visualizations are also interactive; when the user hovers over certain elements on the screen, information about those elements is highlighted. We'll build our initial visualization in just a couple lines of code using *Plotly Express*, a subset of Plotly that focuses on generating plots with as

little code as possible. Once we know our plot is correct, we'll customize the output just as we did with Matplotlib.

En este proyecto, analizaremos los resultados de tirar dados. Cuando lanzas un dado normal de seis caras, tienes la misma probabilidad de sacar cualquiera de los números del 1 al 6. Sin embargo, cuando utilizas dos dados, tienes más probabilidades de sacar ciertos números que otros. Intentaremos determinar qué números tienen más probabilidades de salir generando un conjunto de datos que represente el lanzamiento de dados. Luego trazaremos los resultados de un gran número de tiradas para determinar qué resultados son más probables que otros.

In this project, we'll analyze the results of rolling dice. When you roll one regular, six-sided die, you have an equal chance of rolling any of the numbers from 1 through 6. However, when you use two dice, you're more likely to roll certain numbers than others. We'll try to determine which numbers are most likely to occur by generating a dataset that represents rolling dice. Then we'll plot the results of a large number of rolls to determine which results are more likely than others.

Este trabajo ayuda a modelar los juegos con dados, pero las ideas centrales también se aplican a los juegos que implican azar de cualquier tipo, como los juegos de cartas. También se relaciona con muchas situaciones del mundo real en las que el azar desempeña un factor importante.

This work helps model games involving dice, but the core ideas also apply to games that involve chance of any kind, such as card games. It also relates to many real-world situations where randomness plays a significant factor.

Instalar Plotly (*Installing Plotly*)

Instala Plotly con pip, igual que hiciste con Matplotlib:

Install Plotly using pip, just as you did for Matplotlib:

```
$ python -m pip install --user plotly
$ python -m pip install --user pandas
```

Plotly Express depende de *pandas* , que es una biblioteca para trabajar eficientemente con datos, así que también tenemos que instalarla. Si utilizaste `python3` u otra cosa al instalar Matplotlib, asegúrate de utilizar el mismo comando aquí.

Plotly Express depends on *pandas* , which is a library for working efficiently with data, so we need to install that as well. If you used `python3` or something else when installing Matplotlib, make sure you use the same command here.

Para ver qué tipo de visualizaciones son posibles con Plotly, visita la galería de tipos de gráficos en <https://plotly.com/python> . Cada ejemplo incluye el código fuente, para que puedas ver cómo Plotly genera las visualizaciones.

To see what kind of visualizations are possible with Plotly, visit the gallery of chart types at <https://plotly.com/python> . Each example includes source code, so you can see how Plotly generates the visualizations.

Creación de la clase troquel (*Creating the Die Class*)

Crearemos la siguiente clase `Die` para simular la tirada de un dado:

We'll create the following `Die` class to simulate the roll of one die:

`dado.py`

`die.py`

```
from random import randint
```

```
class Die:
    """A class representing a single die."""
```

```
❶ def __init__(self, num_sides=6):
    """Assume a six-sided die."""
```

```
self.num_sides = num_sides
```

```
def roll(self):
```

```
    """Return a random value between 1 and number of sides."""
```

```
    ❷    return randint(1, self.num_sides)
```

El método `__init__()` toma un argumento opcional ❶. Con la clase `Die`, cuando se crea una instancia de nuestro dado, el número de caras será seis si no se incluye ningún argumento. Si se incluye un argumento `is`, ese valor establecerá el número de caras del dado. (Los dados se denominan según su número de caras: un dado de seis caras es un D6, un dado de ocho caras es un D8, y así sucesivamente).

The `__init__()` method takes one optional argument ❶. With the `Die` class, when an instance of our die is created, the number of sides will be six if no argument is included. If an argument is included, that value will set the number of sides on the die. (Dice are named for their number of sides: a six-sided die is a D6, an eight-sided die is a D8, and so on.)

El método `roll()` utiliza la función `randint()` para devolver un número aleatorio entre 1 y el número de caras ❷. Esta función puede devolver el valor inicial (1), el valor final (`num_sides`), o cualquier número entero entre ambos.

The `roll()` method uses the `randint()` function to return a random number between 1 and the number of sides ❷. This function can return the starting value (1), the ending value (`num_sides`), or any integer between the two.

Lanzar el dado (*Rolling the Die*)

Antes de crear una visualización basada en la clase `Die`, vamos a lanzar un D6, imprimir los resultados y comprobar que éstos parecen razonables:

Before creating a visualization based on the `Die` class, let's roll a D6, print the results, and check that the results look reasonable:

die_visual.py

die_visual.py

```
from die import Die

# Create a D6.
❶ die = Die()

# Make some rolls, and store results in a list.
results = []
❷ for roll_num in range(100):
    result = die.roll()
    results.append(result)

print(results)
```

Creemos una instancia de `Die` con los seis lados por defecto ❶. A continuación, lanzamos el dado 100 veces ❷ y almacenamos el resultado de cada tirada en la lista `results`. Aquí tienes un ejemplo de resultados:

We create an instance of `Die` with the default six sides ❶. Then we roll the die 100 times ❷ and store the result of each roll in the list `results`. Here's a sample set of results:

```
[4, 6, 5, 6, 1, 5, 6, 3, 5, 3, 5, 3, 2, 2, 1, 3, 1, 5, 3, 6, 3, 6, 5, 4, 1, 1, 4, 2, 3, 6, 4, 2, 6, 4, 1, 3, 2, 5, 6,
3, 6, 2, 1, 1, 3, 4, 1, 4, 3, 5, 1, 4, 5, 5, 2, 3, 3, 1, 2, 3, 5, 6, 2, 5, 6, 1, 3, 2, 1, 1, 1, 6, 5, 5, 2, 2, 6, 4, 1,
4, 5, 1, 1, 1, 4, 5, 3, 3, 1, 3, 5, 4, 5, 6, 5, 4, 1, 5, 1, 2]
```

Un examen rápido de estos resultados muestra que la clase `Die` parece funcionar. Vemos los valores 1 y 6, por lo que sabemos que se están devolviendo los valores más pequeño y más grande posibles, y como no vemos 0 ni 7, sabemos que todos los resultados están en el rango apropiado. También vemos cada número del 1 al 6, lo que indica que están representados todos los resultados posibles. Determinemos exactamente cuántas veces aparece cada número.

A quick scan of these results shows that the `Die` class seems to be working. We see the values 1 and 6, so we know the smallest and largest possible values are being returned, and because we don't see 0 or 7, we know all the results are in the appropriate range. We also see each number from 1 through 6, which indicates that all possible outcomes are represented. Let's determine exactly how many times each number appears.

Analizar los resultados (*Analyzing the Results*)

Analizaremos los resultados de tirar un D6 contando cuántas veces sale cada número:

We'll analyze the results of rolling one D6 by counting how many times we roll each number:

die_visual.py

die_visual.py

```
--snip--
# Make some rolls, and store results in a list.
results = []
❶ for roll_num in range(1000):
    result = die.roll()
    results.append(result)

# Analyze the results.
frequencies = []
❷ poss_results = range(1, die.num_sides+1)
for value in poss_results:
    ❸ frequency = results.count(value)
    ❹ frequencies.append(frequency)

print(frequencies)
```

Como ya no vamos a imprimir los resultados, podemos aumentar el número de tiradas simuladas a 1000 ❶. Para analizar las tiradas, creamos la lista vacía `frequencies` para almacenar el número de veces que sale cada valor. A continuación, generamos todos los resultados posibles que podríamos obtener; en este ejemplo, son todos los números desde 1 hasta el número de caras que tenga `die` ❷. Recorremos en bucle los valores posibles, contamos cuántas veces aparece cada número en `results` ❸, y luego añadimos este valor a `frequencies` ❹. Imprimimos esta lista antes de hacer una visualización:

Because we're no longer printing the results, we can increase the number of simulated rolls to 1000 ❶. To analyze the rolls, we create the empty list `frequencies` to store the number of times each value is rolled. We then generate all the possible results we could get; in this example, that's all the numbers from 1 to however many sides `die` has ❷. We loop through the possible values, count how many times each number appears in `results` ❸, and then append this value to `frequencies` ❹. We print this list before making a visualization:

```
[155, 167, 168, 170, 159, 181]
```

Estos resultados parecen razonables: vemos seis frecuencias, una para cada número posible al tirar un D6. También vemos que ninguna frecuencia es significativamente mayor que otra. Ahora vamos a visualizar estos resultados.

These results look reasonable: we see six frequencies, one for each possible number when you roll a D6. We also see that no frequency is significantly higher than any other. Now let's visualize these results.

Hacer un histograma *(Making a Histogram)*

Ahora que tenemos los datos que queremos, podemos generar una visualización en sólo un par de líneas de código utilizando Plotly Express:

Now that we have the data we want, we can generate a visualization in just a couple lines of code using Plotly Express:

die_visual.py

die_visual.py

```
import plotly.express as px
```

```
from die import Die
```

```
--snip--
```

```
for value in poss_results:
    frequency = results.count(value)
    frequencies.append(frequency)

# Visualize the results.
fig = px.bar(x=poss_results, y=frequencies)
fig.show()
```

Primero importamos el módulo `plotly.express` , utilizando el alias convencional `px` . A continuación, utilizamos la función `px.bar()` para crear un gráfico de barras. En el uso más sencillo de esta función, sólo necesitamos pasar un conjunto de valores `x`- y un conjunto de valores `y`- . Aquí, los valores `x` son los posibles resultados de lanzar un solo dado, y los valores `y` son las frecuencias de cada resultado posible.

We first import the `plotly.express` module, using the conventional alias `px` . We then use the `px.bar()` function to create a bar graph. In the simplest use of this function, we only need to pass a set of `x`- values and a set of `y`- values. Here the `x` -values are the possible results from rolling a single die, and the `y` -values are the frequencies for each possible result.

La última línea llama a `fig.show()` , que indica a Plotly que muestre el gráfico resultante como un archivo HTML y lo abra en una nueva pestaña del navegador. El resultado se muestra en [la Figura 15-12](#) .

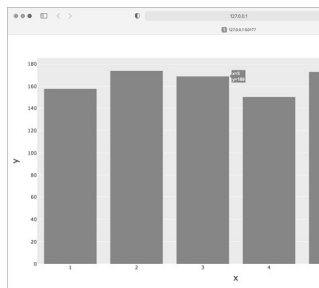
The final line calls `fig.show()` , which tells Plotly to render the resulting chart as an HTML file and open that file in a new browser tab. The result is shown in [Figure 15-12](#) .

Este es un gráfico muy simple, y desde luego no está completo. Pero así es exactamente como debe utilizarse Plotly Express: escribes un par de líneas de código, miras el gráfico y te aseguras de que representa los datos como tú quieres. Si te gusta lo que ves, puedes pasar a personalizar elementos del gráfico como etiquetas y estilos. Pero si quieres explorar otros tipos de gráficos posibles, puedes hacerlo ahora, sin haber dedicado tiempo extra a las tareas de personalización. No dudes en probarlo ahora cambiando `px.bar()` por algo como `px.scatter()` o `px.line()` . Puedes encontrar una lista completa de los tipos de gráficos disponibles en <https://plotly.com/python/plotly-express> .

This is a really simple chart, and it's certainly not complete. But this is exactly how Plotly Express is meant to be used; you write a couple lines of code, look at the plot, and make sure it represents the data the way you want it to. If you like what you see, you can move on to customizing elements of the chart such as labels and styles. But if you want to explore other possible chart types, you can do so now, without having spent extra time on customization work. Feel free to try this now by changing `px.bar()` to something like `px.scatter()` or `px.line()`. You can find a full list of available chart types at <https://plotly.com/python/plotly-express>.

Este gráfico es dinámico e interactivo. Si cambias el tamaño de la ventana de tu navegador, el gráfico cambiará de tamaño para adaptarse al espacio disponible. Si pasas el ratón por encima de cualquiera de las barras, verás una ventana emergente que resalta los datos específicos relacionados con esa barra.

This chart is dynamic and interactive. If you change the size of your browser window, the chart will resize to match the available space. If you hover over any of the bars, you'll see a pop-up highlighting the specific data related to that bar.



DESCRIPCIÓN DE LA IMAGEN (En español):

La imagen muestra un gráfico de barras con un fondo en blanco. En el eje horizontal, etiquetado como "X", se encuentran los números del 1 al 6. En el eje vertical, etiquetado como "Y", se presentan valores numéricos agrupados de 0 a 180 en intervalos. Cada barra está numerada en la base con un número correspondiente de 1 a 6. Hay un cuadro emergente encima de la segunda barra que muestra información con "x=3 y=160". El diseño incluye una

barra superior que
simula controles de
ventana, como
minimizar y cerrar.

Figura 15-12 : Gráfico inicial producido por Plotly Express

Figure 15-12 : The initial plot produced by Plotly Express

Personalizar el gráfico (Customizing the Plot)

Ahora que sabemos que tenemos el tipo de gráfico correcto y que nuestros datos se representan con precisión, podemos centrarnos en añadir las etiquetas y estilos adecuados para el gráfico.

Now that we know we have the correct kind of plot and our data is being represented accurately, we can focus on adding the appropriate labels and styles for the chart.

La primera forma de personalizar un gráfico con Plotly es utilizar algunos parámetros opcionales en la llamada inicial que genera el gráfico, en este caso, `px.bar()` . He aquí cómo añadir un título general y una etiqueta para cada eje:

The first way to customize a plot with Plotly is to use some optional parameters in the initial call that generates the plot, in this case, `px.bar()` . Here's how to add an overall title and a label for each axis:

die_visual.py

die_visual.py

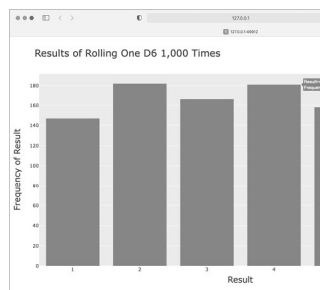
```
--snip--
# Visualize the results.
❶ title = "Results of Rolling One D6 1,000 Times"
❷ labels = {'x': 'Result', 'y': 'Frequency of Result'}
fig = px.bar(x=pos_results, y=frequencies, title=title, labels=labels)
fig.show()
```

Primero definimos el título que queremos, aquí asignado a `title` ❶. Para definir las etiquetas de los ejes, escribimos un diccionario ❷. Las claves del diccionario se refieren a las etiquetas que queremos personalizar, y los valores son las etiquetas personalizadas que queremos utilizar. Aquí damos al eje `x` la etiqueta `Result` y al eje `y` la etiqueta `Frequency of Result`. La llamada a `px.bar()` incluye ahora los argumentos opcionales `title` y `labels`.

We first define the title that we want, here assigned to `title` ❶. To define axis labels, we write a dictionary ❷. The keys in the dictionary refer to the labels we want to customize, and the values are the custom labels we want to use. Here we give the `x`-axis the label `Result` and the `y`-axis the label `Frequency of Result`. The call to `px.bar()` now includes the optional arguments `title` and `labels`.

Ahora, cuando se genera el gráfico, incluye un título apropiado y una etiqueta para cada eje, como se muestra en la [Figura 15-13](#).

Now when the plot is generated it includes an appropriate title and a label for each axis, as shown in [Figure 15-13](#).



DESCRIPCIÓN DE LA
IMAGEN (En español):

Resultados de lanzar
un D6 una vez 1,000
veces Frecuencia del
resultado Resultado 4
(Resultado) Frecuencia
del resultado: 181

Figura 15-13 : Gráfico de barras sencillo creado con Plotly

Figure 15-13 : A simple bar chart created with Plotly

[Tirar dos dados \(Rolling Two Dice\)](#)

Al lanzar dos dados se obtienen números más grandes y una distribución diferente de los resultados. Modifiquemos nuestro código para crear dos dados D6 que simulen la forma en que lanzamos un par de dados. Cada vez que lancemos el par, sumaremos los dos números (uno de cada dado) y almacenaremos la suma en `results`. Guarda una copia de `die_visual.py` como `dice_visual.py` y realiza los siguientes cambios:

Rolling two dice results in larger numbers and a different distribution of results. Let's modify our code to create two D6 dice to simulate the way we roll a pair of dice. Each time we roll the pair, we'll add the two numbers (one from each die) and store the sum in results . Save a copy of *die_visual.py* as *dice_visual.py* and make the following changes:

dice_visual.py

dice_visual.py

```
import plotly.express as px

from die import Die

# Create two D6 dice.
die_1 = Die()
die_2 = Die()

# Make some rolls, and store results in a list.
results = []
for roll_num in range(1000):
    ❶ result = die_1.roll() + die_2.roll()
    results.append(result)

# Analyze the results.
frequencies = []
    ❷ max_result = die_1.num_sides + die_2.num_sides
    ❸ poss_results = range(2, max_result+1)
for value in poss_results:
    frequency = results.count(value)
    frequencies.append(frequency)

# Visualize the results.
title = "Results of Rolling Two D6 Dice 1,000 Times"
labels = {'x': 'Result', 'y': 'Frequency of Result'}
fig = px.bar(x=poss_results, y=frequencies, title=title, labels=labels)
fig.show()
```

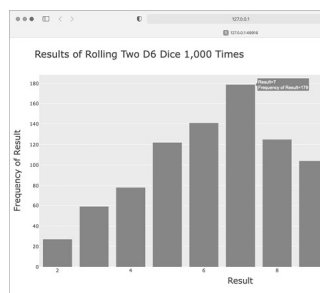
Después de crear dos instancias de `Die` , tiramos los dados y calculamos la suma de los dos dados de cada tirada ❶ . El menor resultado posible (2) es la suma del menor número de cada dado. El mayor resultado posible (12) es la suma del mayor número de cada dado, que

asignamos a `max_result` ❷ . La variable `max_result` hace que el código para generar `poss_results` sea mucho más fácil de leer ❸ . Podríamos haber escrito `range(2, 13)` , pero esto sólo funcionaría para dos dados D6. Al modelar situaciones del mundo real, es mejor escribir un código que pueda modelar fácilmente diversas situaciones. Este código nos permite simular el lanzamiento de un par de dados con cualquier número de caras.

After creating two instances of `Die` , we roll the dice and calculate the sum of the two dice for each roll ❶ . The smallest possible result (2) is the sum of the smallest number on each die. The largest possible result (12) is the sum of the largest number on each die, which we assign to `max_result` ❷ . The variable `max_result` makes the code for generating `poss_results` much easier to read ❸ . We could have written `range(2, 13)` , but this would work only for two D6 dice. When modeling real-world situations, it's best to write code that can easily model a variety of situations. This code allows us to simulate rolling a pair of dice with any number of sides.

Después de ejecutar este código, deberías ver un gráfico parecido al de [la Figura 15-14](#) .

After running this code, you should see a chart that looks like [Figure 15-14](#) .



DESCRIPCIÓN DE LA
IMAGEN (En español):

Resultados de lanzar
dos dados D6 1000
veces Frecuencia del
resultado Resultado

Figura 15-14 : Resultados simulados de lanzar dos dados de seis caras 1.000 veces

Figure 15-14 : Simulated results of rolling two six-sided dice 1,000 times

Este gráfico muestra la distribución aproximada de resultados que es probable que obtengas al lanzar un par de dados D6. Como puedes ver, lo menos probable es que saques un 2 o un 12 y lo más probable es que saques un 7. Esto ocurre porque hay seis formas de sacar un 7: 1 y 6, 2 y 5, 3 y 4, 4 y 3, 5 y 2, y 6 y 1.

This graph shows the approximate distribution of results you're likely to get when you roll a pair of D6 dice. As you can see, you're least likely to roll a 2 or a 12 and most likely to roll

a 7. This happens because there are six ways to roll a 7: 1 and 6, 2 and 5, 3 and 4, 4 and 3, 5 and 2, and 6 and 1.

Otras personalizaciones (*Further Customizations*)

Hay un problema que debemos resolver con la trama que acabamos de generar. Ahora que hay 11 barras, la configuración por defecto del eje x deja algunas barras sin etiquetar. Aunque la configuración por defecto funciona bien para la mayoría de las visualizaciones, este gráfico tendría mejor aspecto con todas las barras etiquetadas.

There's one issue that we should address with the plot we just generated. Now that there are 11 bars, the default layout settings for the x-axis leave some of the bars unlabeled. While the default settings work well for most visualizations, this chart would look better with all of the bars labeled.

Plotly tiene un método `update_layout()` que puede utilizarse para realizar una gran variedad de actualizaciones en una figura después de haberla creado. He aquí cómo decirle a Plotly que dé a cada barra su propia etiqueta:

Plotly has an `update_layout()` method that can be used to make a wide variety of updates to a figure after it's been created. Here's how to tell Plotly to give each bar its own label:

dice_visual.py

dice_visual.py

```
--snip--
fig = px.bar(x=poss_results, y=frequencies, title=title, labels=labels)

# Further customize chart.
fig.update_layout(xaxis_dtick=1)

fig.show()
```

El método `update_layout()` actúa sobre el objeto `fig` , que representa el gráfico general. Aquí utilizamos el argumento `xaxis_dtick` , que especifica la distancia entre las marcas de graduación en el eje `x` . Fijamos ese espaciado en `1` , para que cada barra esté etiquetada. Cuando vuelvas a ejecutar `dice_visual.py` , deberías ver una etiqueta en cada barra.

The `update_layout()` method acts on the `fig` object, which represents the overall chart. Here we use the `xaxis_dtick` argument, which specifies the distance between tick marks on the `x`-axis. We set that spacing to `1` , so that every bar is labeled. When you run `dice_visual.py` again, you should see a label on each bar.

Lanzar dados de distintos tamaños (*Rolling Dice of Different Sizes*)

Vamos a crear un dado de seis caras y otro de diez caras, y a ver qué ocurre cuando los lanzamos 50.000 veces:

Let's create a six-sided die and a ten-sided die, and see what happens when we roll them 50,000 times:

`dado_visual_d6d10.py`

`dice_visual_d6d10.py`

```
import plotly.express as px

from die import Die

# Create a D6 and a D10.
die_1 = Die()
❶ die_2 = Die(10)

# Make some rolls, and store results in a list.
results = []
for roll_num in range(50_000):
    result = die_1.roll() + die_2.roll()
    results.append(result)
```

```
# Analyze the results.  
--snip--
```

```
# Visualize the results.
```

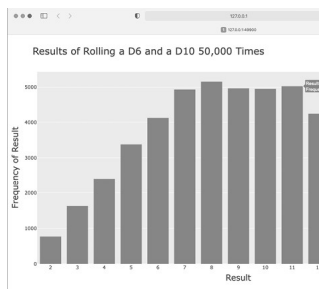
```
❷ title = "Results of Rolling a D6 and a D10 50,000 Times"  
labels = {'x': 'Result', 'y': 'Frequency of Result'}  
--snip--
```

Para crear un D10, pasamos el argumento 10 al crear la segunda instancia Die ❶ y cambiamos el primer bucle para simular 50.000 tiradas en lugar de 1.000. También cambiamos el título del gráfico ❷ .

To make a D10, we pass the argument 10 when creating the second Die instance ❶ and change the first loop to simulate 50,000 rolls instead of 1,000. We change the title of the graph as well ❷ .

La Figura 15-15 muestra el gráfico resultante. En lugar de un resultado más probable, hay cinco resultados de este tipo. Esto ocurre porque sigue habiendo una sola forma de sacar el valor más pequeño (1 y 1) y el valor más grande (6 y 10), pero el dado más pequeño limita el número de formas en que puedes generar los números intermedios. Hay seis formas de sacar un 7, 8, 9, 10 u 11, que son los resultados más comunes, y tienes las mismas probabilidades de sacar cualquiera de ellos.

Figure 15-15 shows the resulting chart. Instead of one most likely result, there are five such results. This happens because there's still only one way to roll the smallest value (1 and 1) and the largest value (6 and 10), but the smaller die limits the number of ways you can generate the middle numbers. There are six ways to roll a 7, 8, 9, 10, or 11, these are the most common results, and you're equally likely to roll any one of them.



DESCRIPCIÓN DE LA
IMAGEN (En español):

Resultados de lanzar
un D6 y un D10 50,000
veces Frecuencia del
resultado Resultado
Frecuencia del
resultado: 3021

Figura 15-15 : Los resultados de lanzar un dado de seis caras y un dado de diez caras 50.000 veces

Figure 15-15 : The results of rolling a six-sided die and a ten-sided die 50,000 times

La posibilidad de utilizar Plotly para modelizar el lanzamiento de dados nos da una libertad considerable para explorar este fenómeno. En sólo unos minutos, puedes simular un enorme número de tiradas utilizando una gran variedad de dados.

Our ability to use Plotly to model the rolling of dice gives us considerable freedom in exploring this phenomenon. In just minutes, you can simulate a tremendous number of rolls using a large variety of dice.

Guardar cifras (Saving Figures)

Cuando tengas una figura que te guste, siempre puedes guardar el gráfico como un archivo HTML a través de tu navegador. Pero también puedes hacerlo mediante programación. Para guardar tu gráfico como un archivo HTML, sustituye la llamada a `fig.show()` por una llamada a `fig.write_html()` :

When you have a figure you like, you can always save the chart as an HTML file through your browser. But you can also do so programmatically. To save your chart as an HTML file, replace the call to `fig.show()` with a call to `fig.write_html()` :

```
fig.write_html('dice_visual_d6d10.xhtml')
```

El método `write_html()` requiere un argumento: el nombre del archivo en el que escribir. Si sólo proporcionas un nombre de archivo, el archivo se guardará en el mismo directorio que el archivo `.py` . También puedes llamar a `write_html()` con un objeto `Path` , y escribir el archivo de salida en cualquier lugar que desees de tu sistema.

The `write_html()` method requires one argument: the name of the file to write to. If you only provide a filename, the file will be saved in the same directory as the `.py` file. You can also call `write_html()` with a `Path` object, and write the output file anywhere you want on your system.

Pruébalo tú mismo (*Try It Yourself*)

15-6. Dos D8: Crea una simulación que muestre lo que ocurre cuando tiras dos dados de ocho caras 1.000 veces. Intenta imaginarte cómo crees que será la visualización antes de ejecutar la simulación, y luego comprueba si tu intuición era correcta. Aumenta gradualmente el número de tiradas hasta que empieces a ver los límites de las capacidades de tu sistema.

15-6. Two D8s: Create a simulation showing what happens when you roll two eight-sided dice 1,000 times. Try to picture what you think the visualization will look like before you run the simulation, then see if your intuition was correct. Gradually increase the number of rolls until you start to see the limits of your system's capabilities.

15-7. Tres dados: Cuando lanzas tres dados D6, el número más pequeño que puedes sacar es 3 y el más grande 18. Crea una visualización que muestre lo que ocurre cuando tiras tres dados D6.

15-7. Three Dice: When you roll three D6 dice, the smallest number you can roll is 3 and the largest number is 18. Create a visualization that shows what happens when you roll three D6 dice.

15-8. Multiplicación: Cuando tiras dos dados, normalmente sumas los dos números para obtener el resultado. Crea una visualización que muestre qué ocurre si, en lugar de eso, multiplicas esos números entre sí.

15-8. Multiplication: When you roll two dice, you usually add the two numbers together to get the result. Create a visualization that shows what happens if you multiply these numbers by each other instead.

15-9. Comprensión de dados: Para mayor claridad, los listados de esta sección utilizan la forma larga de los bucles `for`. Si te sientes cómodo utilizando comprensiones de listas, intenta escribir una comprensión para uno o los dos bucles de cada uno de estos programas.

15-9. Die Comprehensions: For clarity, the listings in this section use the long form of for loops. If you're comfortable using list comprehensions, try writing a comprehension for one or both of the loops in each of these programs.

15-10. Practicando con ambas bibliotecas: Intenta utilizar Matplotlib para hacer una visualización de una tirada de dados, y utiliza Plotly para hacer la visualización de un paseo aleatorio. (Tendrás que consultar la documentación de cada biblioteca para completar este ejercicio).

15-10. Practicing with Both Libraries: Try using Matplotlib to make a die-rolling visualization, and use Plotly to make the visualization for a random walk. (You'll need to consult the documentation for each library to complete this exercise.)

Resumen (Summary)

En este capítulo, has aprendido a generar conjuntos de datos y a crear visualizaciones de esos datos. Creaste gráficos sencillos con Matplotlib y utilizaste un gráfico de dispersión para explorar los paseos aleatorios. También creaste un histograma con Plotly, y lo utilizaste para explorar los resultados de lanzar dados de diferentes tamaños.

In this chapter, you learned to generate datasets and create visualizations of that data. You created simple plots with Matplotlib and used a scatter plot to explore random walks. You also created a histogram with Plotly, and used it to explore the results of rolling dice of different sizes.

Generar tus propios conjuntos de datos con código es una forma interesante y potente de modelar y explorar una amplia variedad de situaciones del mundo real. A medida que sigas trabajando en los proyectos de visualización de datos que siguen, estate atento a las situaciones que puedas modelar con código. Observa las visualizaciones que ves en los medios de comunicación, y mira si puedes identificar las que se generaron utilizando métodos similares a los que estás aprendiendo en estos proyectos.

Generating your own datasets with code is an interesting and powerful way to model and explore a wide variety of real-world situations. As you continue to work through the data visualization projects that follow, keep an eye out for situations you might be able to model

with code. Look at the visualizations you see in news media, and see if you can identify those that were generated using methods similar to the ones you're learning in these projects.

En el Capítulo 16 , descargarás datos de fuentes online y seguirás utilizando Matplotlib y Plotly para explorar esos datos.

In Chapter 16 , you'll download data from online sources and continue to use Matplotlib and Plotly to explore that data.

Capítulo 16: Descarga de datos (Chapter 16: Downloading Data)

16

Descargar datos (16 Downloading Data)



DESCRIPCIÓN DE LA
IMAGEN (En español):

La imagen muestra una ilustración de una serpiente enroscada alrededor de un potente motor a reacción. La serpiente tiene un aspecto caricaturesco con una expresión decidida en su rostro, destacando por sus ojos grandes y enfocados. El motor está detallado con paletas visibles, y el diseño general es en blanco y negro, con un fondo gris que resalta las figuras principales. No hay texto presente en la imagen.

En este capítulo, descargarás conjuntos de datos de fuentes online y crearás visualizaciones de trabajo de esos datos. Puedes encontrar una increíble variedad de datos en Internet, muchos de los cuales no se han examinado a fondo. La capacidad de analizar estos datos te permite descubrir patrones y conexiones que nadie más ha encontrado.

In this chapter, you'll download datasets from online sources and create working visualizations of that data. You can find an incredible variety of data online, much of which hasn't been examined thoroughly. The ability to analyze this data allows you to discover patterns and connections that no one else has found.

Accederemos y visualizaremos datos almacenados en dos formatos de datos comunes: CSV y JSON. Utilizaremos el módulo `csv` de Python para procesar los datos meteorológicos

almacenados en el formato CSV y analizar las temperaturas altas y bajas a lo largo del tiempo en dos lugares diferentes. A continuación, utilizaremos Matplotlib para generar un gráfico basado en nuestros datos descargados para mostrar las variaciones de temperatura en dos entornos distintos: Sitka, Alaska, y el Valle de la Muerte, California. Más adelante en el capítulo, utilizaremos el módulo `json` para acceder a los datos de terremotos almacenados en formato GeoJSON y utilizaremos Plotly para dibujar un mapa del mundo que muestre las ubicaciones y magnitudes de los terremotos recientes.

We'll access and visualize data stored in two common data formats: CSV and JSON. We'll use Python's `csv` module to process weather data stored in the CSV format and analyze high and low temperatures over time in two different locations. We'll then use Matplotlib to generate a chart based on our downloaded data to display variations in temperature in two dissimilar environments: Sitka, Alaska, and Death Valley, California. Later in the chapter, we'll use the `json` module to access earthquake data stored in the GeoJSON format and use Plotly to draw a world map showing the locations and magnitudes of recent earthquakes.

Al final de este capítulo, estarás preparado para trabajar con varios tipos de conjuntos de datos en diferentes formatos, y tendrás un conocimiento más profundo de cómo construir visualizaciones complejas. Ser capaz de acceder y visualizar datos en línea es esencial para trabajar con una amplia variedad de conjuntos de datos del mundo real.

By the end of this chapter, you'll be prepared to work with various types of datasets in different formats, and you'll have a deeper understanding of how to build complex visualizations. Being able to access and visualize online data is essential to working with a wide variety of real-world datasets.

El formato de archivo CSV (*The CSV File Format*)

Una forma sencilla de almacenar datos en un archivo de texto es escribir los datos como una serie de valores separados por comas, denominados *comma-separated values*. Los archivos resultantes son archivos CSV. Por ejemplo, aquí tienes un trozo de datos meteorológicos en formato CSV:

One simple way to store data in a text file is to write the data as a series of values separated by commas, called *comma-separated values*. The resulting files are CSV files. For example, here's a chunk of weather data in CSV format:

"USW00025333","SITKA AIRPORT, AK US","2021-01-01","44","40"

Este es un extracto de los datos meteorológicos del 1 de enero de 2021, en Sitka, Alaska. Incluye las temperaturas máxima y mínima del día, así como otras mediciones de ese día. Los archivos CSV pueden resultar tediosos de leer para los humanos, pero los programas pueden procesar y extraer información de ellos con rapidez y precisión.

This is an excerpt of weather data from January 1, 2021, in Sitka, Alaska. It includes the day's high and low temperatures, as well as a number of other measurements from that day. CSV files can be tedious for humans to read, but programs can process and extract information from them quickly and accurately.

Empezaremos con un pequeño conjunto de datos meteorológicos en formato CSV registrados en Sitka; está disponible en los recursos de este libro en https://ehmatthes.github.io/pcc_3e. Crea una carpeta llamada *weather_data* dentro de la carpeta donde guardes los programas de este capítulo. Copia el archivo *sitka_weather_07-2021_simple.csv* en esta nueva carpeta. (Después de descargar los recursos de este libro, tendrás todos los archivos que necesitas para este proyecto).

We'll begin with a small set of CSV-formatted weather data recorded in Sitka; it is available in this book's resources at https://ehmatthes.github.io/pcc_3e. Make a folder called *weather_data* inside the folder where you're saving this chapter's programs. Copy the file *sitka_weather_07-2021_simple.csv* into this new folder. (After you download this book's resources, you'll have all the files you need for this project.)

Nota (Note)

Los datos meteorológicos de este proyecto se descargaron originalmente de <https://ncdc.noaa.gov/cdo-web>.

The weather data in this project was originally downloaded from <https://ncdc.noaa.gov/cdo-web>.

Análisis de las cabeceras del archivo CSV (*Parsing the CSV File Headers*)

El módulo `csv` de la biblioteca estándar de Python analiza las líneas de un archivo CSV y nos permite extraer rápidamente los valores que nos interesan. Empecemos examinando la primera línea del archivo, que contiene una serie de cabeceras para los datos. Estas cabeceras nos indican qué tipo de información contienen los datos:

Python's `csv` module in the standard library parses the lines in a CSV file and allows us to quickly extract the values we're interested in. Let's start by examining the first line of the file, which contains a series of headers for the data. These headers tell us what kind of information the data holds:

sitka_altos.py

sitka_highs.py

```
from pathlib import Path
import csv
```

```
❶ path = Path('weather_data/sitka_weather_07-2021_simple.csv')
lines = path.read_text().splitlines()
```

```
❷ reader = csv.reader(lines)
❸ header_row = next(reader)
print(header_row)
```

Primero importamos `Path` y el módulo `csv`. A continuación, construimos un objeto `Path` que busca en la carpeta `weather_data`, y apunta al archivo de datos meteorológicos específico con el que queremos trabajar ❶. Leemos el archivo y encadenamos el método `splitlines()` para obtener una lista de todas las líneas del archivo, que asignamos a `lines`.

We first import `Path` and the `csv` module. We then build a `Path` object that looks in the `weather_data` folder, and points to the specific weather data file we want to work with ❶. We read the file and chain the `splitlines()` method to get a list of all lines in the file, which we assign to `lines`.

A continuación, construimos un objeto `reader` ❷ . Éste es un objeto que puede utilizarse para analizar cada línea del archivo. Para crear un objeto lector, llama a la función `csv.reader()` y pásale la lista de líneas del archivo CSV.

Next, we build a `reader` object ❷ . This is an object that can be used to parse each line in the file. To make a reader object, call the function `csv.reader()` and pass it the list of lines from the CSV file.

Cuando se le da un objeto `reader` , la función `next()` devuelve la siguiente línea del archivo, empezando por el principio del archivo. Aquí llamamos a `next()` una sola vez, por lo que obtenemos la primera línea del archivo, que contiene las cabeceras del archivo ❸ . Asignamos los datos devueltos a `header_row` . Como puedes ver, `header_row` contiene cabeceras significativas, relacionadas con el tiempo, que nos indican qué información contiene cada línea de datos:

When given a `reader` object, the `next()` function returns the next line in the file, starting from the beginning of the file. Here we call `next()` only once, so we get the first line of the file, which contains the file headers ❸ . We assign the data that's returned to `header_row` . As you can see, `header_row` contains meaningful, weather-related headers that tell us what information each line of data holds:

```
['STATION', 'NAME', 'DATE', 'TAVG', 'TMAX', 'TMIN']
```

El objeto `reader` procesa la primera línea de valores separados por comas del archivo y almacena cada valor como un elemento de una lista. La cabecera `STATION` representa el código de la estación meteorológica que registró estos datos. La posición de esta cabecera nos indica que el primer valor de cada línea será el código de la estación meteorológica. La cabecera `NAME` indica que el segundo valor de cada línea es el nombre de la estación meteorológica que realizó la grabación. El resto de las cabeceras especifican qué tipo de información se registró en cada lectura. Los datos que más nos interesan por ahora son la fecha (`DATE`), la temperatura alta (`TMAX`) y la temperatura baja (`TMIN`). Se trata de un conjunto de datos sencillo que sólo contiene datos relacionados con la temperatura. Cuando descargues tus propios datos meteorológicos, puedes optar por incluir otra serie de mediciones relacionadas con la velocidad y dirección del viento, y datos sobre precipitaciones.

The `reader` object processes the first line of comma-separated values in the file and stores each value as an item in a list. The header `STATION` represents the code for the weather station that recorded this data. The position of this header tells us that the first value in each line will be the weather station code. The `NAME` header indicates that the second value in each line is the name of the weather station that made the recording. The rest of the headers specify what kinds of information were recorded in each reading. The data we're most interested in for now are the date (`DATE`), the high temperature (`TMAX`), and the low temperature (`TMIN`). This is a simple dataset that contains only temperature-related data. When you download your own weather data, you can choose to include a number of other measurements relating to wind speed, wind direction, and precipitation data.

Imprimir las cabeceras y sus posiciones *(Printing the Headers and Their Positions)*

Para facilitar la comprensión de los datos de cabecera del archivo, vamos a imprimir cada cabecera y su posición en la lista:

To make it easier to understand the file header data, let's print each header and its position in the list:

sitka_cabeceras.py

sitka_highs.py

```
--snip--
reader = csv.reader(lines)
header_row = next(reader)

for index, column_header in enumerate(header_row):
    print(index, column_header)
```

La función `enumerate()` devuelve tanto el índice de cada elemento como el valor de cada elemento al recorrer una lista. (Observa que hemos eliminado la línea `print(header_row)` en favor de esta versión más detallada).

The `enumerate()` function returns both the index of each item and the value of each item as you loop through a list. (Note that we've removed the line `print(header_row)` in favor of this more detailed version.)

Aquí tienes la salida mostrando el índice de cada encabezado:

Here's the output showing the index of each header:

```
0 STATION
1 NAME
2 DATE
3 TAVG
4 TMAX
5 TMIN
```

Podemos ver que las fechas y sus altas temperaturas se almacenan en las columnas 2 y 4. Para explorar estos datos, procesaremos cada fila de datos en *sitka_weather_07-2021_simple.csv* y extraeremos los valores con los índices 2 y 4.

We can see that the dates and their high temperatures are stored in columns 2 and 4. To explore this data, we'll process each row of data in *sitka_weather_07-2021_simple.csv* and extract the values with the indexes 2 and 4.

Extraer y leer datos (*Extracting and Reading Data*)

Ahora que sabemos qué columnas de datos necesitamos, vamos a leer algunos de esos datos. En primer lugar, leeremos la temperatura máxima de cada día:

Now that we know which columns of data we need, let's read in some of that data. First, we'll read in the high temperature for each day:

`sitka_temperaturas_altas.py`

`sitka_highs.py`

```
--snip--
reader = csv.reader(lines)
header_row = next(reader)

# Extract high temperatures.
❶ highs = []
❷ for row in reader:
❸     high = int(row[4])
    highs.append(high)

print(highs)
```

Hacemos una lista vacía llamada `highs` ❶ y luego recorremos en bucle las filas restantes del archivo ❷. El objeto `reader` continúa desde donde lo dejó en el archivo CSV y devuelve automáticamente cada línea siguiente a su posición actual. Como ya hemos leído la fila de cabecera, el bucle comenzará en la segunda línea, donde empiezan los datos reales. En cada pasada por el bucle, extraemos los datos del índice 4, correspondiente a la cabecera `TMAX`, y los asignamos a la variable `high` ❸. Utilizamos la función `int()` para convertir los datos, que se almacenan como una cadena, a un formato numérico para que podamos utilizarlos. A continuación, añadimos este valor a `highs`.

We make an empty list called `highs` ❶ and then loop through the remaining rows in the file ❷. The `reader` object continues from where it left off in the CSV file and automatically returns each line following its current position. Because we've already read the header row, the loop will begin at the second line where the actual data begins. On each pass through the loop we pull the data from index 4, corresponding to the header `TMAX`, and assign it to the variable `high` ❸. We use the `int()` function to convert the data, which is stored as a string, to a numerical format so we can use it. We then append this value to `highs`.

El siguiente listado muestra los datos almacenados ahora en `highs` :

The following listing shows the data now stored in `highs` :

```
[61, 60, 66, 60, 65, 59, 58, 58, 57, 60, 60, 60, 57, 58, 60, 61, 63, 63, 70, 64, 59, 63, 61, 58, 59, 64,
62, 70, 70, 73, 66]
```


Hemos extraído la temperatura alta para cada fecha y almacenado cada valor en una lista. Ahora vamos a crear una visualización de estos datos.

We've extracted the high temperature for each date and stored each value in a list. Now let's create a visualization of this data.

Trazar los datos en un gráfico de temperatura (*Plotting Data in a Temperature Chart*)

Para visualizar los datos de temperatura que tenemos, primero crearemos un gráfico simple de las máximas diarias utilizando Matplotlib, como se muestra aquí:

To visualize the temperature data we have, we'll first create a simple plot of the daily highs using Matplotlib, as shown here:

sitka_altas.py

sitka_highs.py

```
from pathlib import Path
import csv

import matplotlib.pyplot as plt

path = Path('weather_data/sitka_weather_07-2021_simple.csv')
lines = path.read_text().splitlines()
--snip--

# Plot the high temperatures.
plt.style.use('seaborn')
fig, ax = plt.subplots()
❶ ax.plot(highs, color='red')

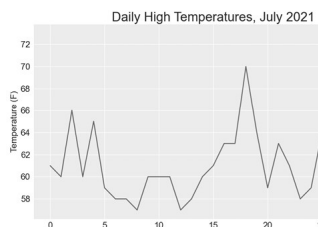
# Format plot.
❷ ax.set_title("Daily High Temperatures, July 2021", fontsize=24)
❸ ax.set_xlabel("", fontsize=16)
ax.set_ylabel("Temperature (F)", fontsize=16)
```

```
ax.tick_params(labelsize=16)
```

```
plt.show()
```

Pasamos la lista de máximas a `plot()` y pasamos a `color='red'` para trazar los puntos en rojo ❶. (Trazaremos las máximas en rojo y las mínimas en azul.) A continuación, especificamos algunos otros detalles de formato, como el título, el tamaño de la fuente y las etiquetas ❷, tal como hicimos en el Capítulo 15. Como aún tenemos que añadir las fechas, no etiquetaremos el eje x, pero `ax.set_xlabel()` sí modifica el tamaño de la fuente para que las etiquetas por defecto sean más legibles ❸. La Figura 16-1 muestra el gráfico resultante: un simple gráfico lineal de las altas temperaturas de julio de 2021 en Sitka, Alaska.

We pass the list of highs to `plot()` and pass `color='red'` to plot the points in red ❶. (We'll plot the highs in red and the lows in blue.) We then specify a few other formatting details, such as the title, font size, and labels ❷, just as we did in Chapter 15. Because we have yet to add the dates, we won't label the x-axis, but `ax.set_xlabel()` does modify the font size to make the default labels more readable ❸. Figure 16-1 shows the resulting plot: a simple line graph of the high temperatures for July 2021 in Sitka, Alaska.



DESCRIPCIÓN DE LA
IMAGEN (En español):

Temperaturas
máximas diarias, julio
2021 Temperatura (°F)
0, 5, 10, 15, 20, 25, 30,
35 58, 60, 62, 64, 66,
68, 70, 72

Figura 16-1 : Gráfico lineal de las temperaturas máximas diarias de julio de 2021 en Sitka, Alaska

Figure 16-1 : A line graph showing daily high temperatures for July 2021 in Sitka, Alaska

El módulo `datetime` (*The `datetime` Module*)

Vamos a añadir fechas a nuestro gráfico para hacerlo más útil. La primera fecha del archivo de datos meteorológicos está en la segunda fila del archivo:

Let's add dates to our graph to make it more useful. The first date from the weather data file is in the second row of the file:

```
"USW00025333","SITKA AIRPORT, AK US","2021-07-01","61","53"
```

Los datos se leerán como una cadena, por lo que necesitamos una forma de convertir la cadena "2021-07-01" en un objeto que represente esta fecha. Podemos construir un objeto que represente el 1 de julio de 2021, utilizando el método `strptime()` del módulo `datetime`. Veamos cómo funciona `strptime()` en una sesión de terminal:

The data will be read in as a string, so we need a way to convert the string "2021-07-01" to an object representing this date. We can construct an object representing July 1, 2021, using the `strptime()` method from the `datetime` module. Let's see how `strptime()` works in a terminal session:

```
>>> from datetime import datetime
>>> first_date = datetime.strptime('2021-07-01', '%Y-%m-%d')
>>> print(first_date)
2021-07-01 00:00:00
```

Primero importamos la clase `datetime` del módulo `datetime`. Luego llamamos al método `strptime()` con la cadena que contiene la fecha que queremos procesar como primer argumento. El segundo argumento indica a Python el formato de la fecha. En este ejemplo, `'%Y-'` indica a Python que busque un año de cuatro dígitos antes del primer guión; `'%m-'` indica un mes de dos dígitos antes del segundo guión; y `'%d'` significa que la última parte de la cadena es el día del mes, del 1 al 31.

We first import the `datetime` class from the `datetime` module. Then we call the method `strptime()` with the string containing the date we want to process as its first argument. The second argument tells Python how the date is formatted. In this example, `'%Y-'` tells Python to look for a four-digit year before the first dash; `'%m-'` indicates a two-digit month before the second dash; and `'%d'` means the last part of the string is the day of the month, from 1 to 31.

El método `strptime()` puede tomar diversos argumentos para determinar cómo interpretar la fecha. [La Tabla 16-1](#) muestra algunos de estos argumentos.

The `strptime()` method can take a variety of arguments to determine how to interpret the date. [Table 16-1](#) shows some of these arguments.

Tabla en español:	Argument	Table in English:	Argument
	%A		%A
	%B		%B
	%m		%m
	%d		%d
	%Y		%Y
	%y		%y
	%H		%H
	%I		%I
	%p		%p
	%M		%M
	%S		%S

Tabla 16-1 : Argumentos de formato de fecha y hora del módulo datetime

Table 16-1 : Date and Time Formatting Arguments from the datetime Module

Trazar fechas (Plotting Dates)

Podemos mejorar nuestro gráfico extrayendo las fechas de las lecturas diarias de temperatura alta, y utilizando estas fechas en el eje x :

We can improve our plot by extracting dates for the daily high temperature readings, and using these dates on the x -axis:

sitka_alta.py

sitka_highs.py

```
from pathlib import Path
import csv
from datetime import datetime

import matplotlib.pyplot as plt
```

```
path = Path('weather_data/sitka_weather_07-2021_simple.csv')
lines = path.read_text().splitlines()
```

```
reader = csv.reader(lines)
header_row = next(reader)
```

```
# Extract dates and high temperatures.
```

```
❶ dates, highs = [], []
for row in reader:
    ❷ current_date = datetime.strptime(row[2], '%Y-%m-%d')
    high = int(row[4])
    dates.append(current_date)
    highs.append(high)
```

```
# Plot the high temperatures.
```

```
plt.style.use('seaborn')
fig, ax = plt.subplots()
❸ ax.plot(dates, highs, color='red')
```

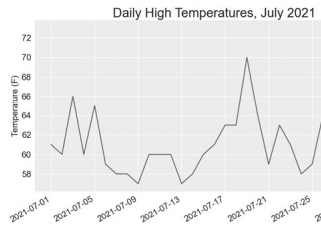
```
# Format plot.
```

```
ax.set_title("Daily High Temperatures, July 2021", fontsize=24)
ax.set_xlabel("", fontsize=16)
❹ fig.autofmt_xdate()
ax.set_ylabel("Temperature (F)", fontsize=16)
ax.tick_params(labelsize=16)
```

```
plt.show()
```

Creamos dos listas vacías para almacenar las fechas y las altas temperaturas del archivo ❶ . A continuación, convertimos los datos que contienen la información de la fecha (`row[2]`) en un objeto `datetime` ❷ y lo añadimos a `dates` . Pasamos las fechas y los valores de las altas temperaturas a `plot()` ❸ . La llamada a `fig.autofmt_xdate()` ❹ dibuja las etiquetas de las fechas en diagonal para evitar que se solapen. La [Figura 16-2](#) muestra el gráfico mejorado.

We create two empty lists to store the dates and high temperatures from the file ❶ . We then convert the data containing the date information (`row[2]`) to a `datetime` object ❷ and append it to `dates` . We pass the dates and the high temperature values to `plot()` ❸ . The call to `fig.autofmt_xdate()` ❹ draws the date labels diagonally to prevent them from overlapping. [Figure 16-2](#) shows the improved graph.



DESCRIPCIÓN DE LA
IMAGEN (En español):

Temperaturas
máximas diarias, julio
2021 Fecha: 2021-07-
01, Temperatura (°F):
64 Fecha: 2021-07-05,
Temperatura (°F): 58
Fecha: 2021-07-09,
Temperatura (°F): 66
Fecha: 2021-07-13,
Temperatura (°F): 58
Fecha: 2021-07-17,
Temperatura (°F): 64
Fecha: 2021-07-21,
Temperatura (°F): 72
Fecha: 2021-07-25,
Temperatura (°F): 60
Fecha: 2021-07-29,
Temperatura (°F): 70
Fecha: 2021-08-01,
Temperatura (°F): 66

Figura 16-2 : El gráfico es más significativo, ahora que tiene fechas en el

Figure 16-2 : The graph is more meaningful, now that it has dates on the x-axis.

eje

x .

Trazar un marco temporal más largo (*Plotting a Longer Timeframe*)

Con nuestro gráfico preparado, vamos a incluir datos adicionales para obtener una imagen más completa del tiempo en Sitka. Copia el archivo *sitka_weather_2021_simple.csv*, que contiene un año completo de datos meteorológicos de Sitka, en la carpeta donde guardas los datos para los programas de este capítulo.

With our graph set up, let's include additional data to get a more complete picture of the weather in Sitka. Copy the file *sitka_weather_2021_simple.csv*, which contains a full year's worth of weather data for Sitka, to the folder where you're storing the data for this chapter's programs.

Ahora podemos generar un gráfico del tiempo de todo el año:

Now we can generate a graph for the entire year's weather:

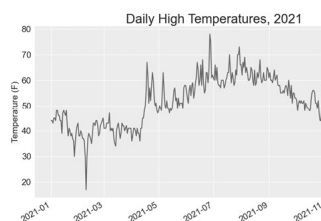
sitka_altas.py

sitka_highs.py

```
--snip--
path = Path('weather_data/sitka_weather_2021_simple.csv')
lines = path.read_text().splitlines()
--snip--
# Format plot.
ax.set_title("Daily High Temperatures, 2021", fontsize=24)
ax.set_xlabel("", fontsize=16)
--snip--
```

Modificamos el nombre del archivo para utilizar el nuevo archivo de datos *sitka_weather_2021_simple.csv*, y actualizamos el título de nuestro gráfico para reflejar el cambio en su contenido. [La Figura 16-3](#) muestra el gráfico resultante.

We modify the filename to use the new data file *sitka_weather_2021_simple.csv*, and we update the title of our plot to reflect the change in its content. [Figure 16-3](#) shows the resulting plot.



DESCRIPCIÓN DE LA
IMAGEN (En español):

Temperaturas
Máximas Diarias, 2021
Temperatura (°F)
2021-01 2021-03
2021-05 2021-07
2021-09 2021-11
2022-01

Figura 16-3 : Un año de datos

Figure 16-3 : A year's worth of data

Trazar una segunda serie de datos (*Plotting a Second Data Series*)

Podemos hacer que nuestro gráfico sea aún más útil incluyendo las bajas temperaturas. Tenemos que extraer las temperaturas bajas del archivo de datos y añadirlas a nuestro

gráfico, como se muestra aquí:

We can make our graph even more useful by including the low temperatures. We need to extract the low temperatures from the data file and then add them to our graph, as shown here:

sitka_altas_bajas.py

sitka_highs_lows.py

```
--snip--
reader = csv.reader(lines)
header_row = next(reader)

# Extract dates, and high and low temperatures.
❶ dates, highs, lows = [], [], []
for row in reader:
    current_date = datetime.strptime(row[2], '%Y-%m-%d')
    high = int(row[4])
    ❷ low = int(row[5])
    dates.append(current_date)
    highs.append(high)
    lows.append(low)

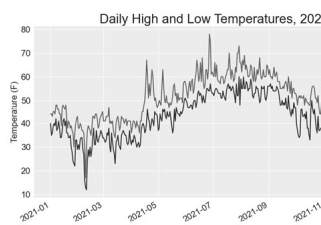
# Plot the high and low temperatures.
plt.style.use('seaborn')
fig, ax = plt.subplots()
ax.plot(dates, highs, color='red')
❸ ax.plot(dates, lows, color='blue')

# Format plot.
❹ ax.set_title("Daily High and Low Temperatures, 2021", fontsize=24)
--snip--
```

Añadimos la lista vacía `lows` para contener las temperaturas bajas ❶, y luego extraemos y almacenamos la temperatura baja de cada fecha a partir de la sexta posición de cada fila (`row[5]`) ❷. Añadimos una llamada a `plot()` para las temperaturas bajas y coloreamos estos valores en azul ❸. Por último, actualizamos el título ❹. [La Figura 16-4](#) muestra el

gráfico resultante.

We add the empty list `lows` to hold low temperatures ❶, and then we extract and store the low temperature for each date from the sixth position in each row (`row[5]`) ❷. We add a call to `plot()` for the low temperatures and color these values blue ❸. Finally, we update the title ❹. [Figure 16-4](#) shows the resulting chart.



DESCRIPCIÓN DE LA
IMAGEN (En español):

Altas y bajas
temperaturas diarias,
2021 Temperatura (°C)
2021-01 2021-03
2021-05 2021-07
2021-09 2021-11
2022-01

Figura 16-4 : Dos series de datos en el mismo gráfico

Figure 16-4 : Two data series on the same plot

Sombrear un área del gráfico (Shading an Area in the Chart)

Tras añadir dos series de datos, ahora podemos examinar el rango de temperaturas de cada día. Vamos a dar un toque final al gráfico utilizando el sombreado para mostrar el rango entre las temperaturas máximas y mínimas de cada día. Para ello, utilizaremos el método `fill_between()`, que toma una serie de valores `x` y dos series de valores `y` y rellena el espacio entre las dos series de valores `y`:

Having added two data series, we can now examine the range of temperatures for each day. Let's add a finishing touch to the graph by using shading to show the range between each day's high and low temperatures. To do so, we'll use the `fill_between()` method, which takes a series of `x`-values and two series of `y`-values and fills the space between the two series of `y`-values:

`sitka_altas_bajas.py`

`sitka_highs_lows.py`

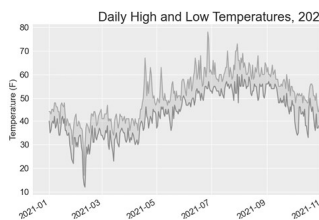
```
--snip--
# Plot the high and low temperatures.
plt.style.use('seaborn')
fig, ax = plt.subplots()
❶ ax.plot(dates, highs, color='red', alpha=0.5)
ax.plot(dates, lows, color='blue', alpha=0.5)
❷ ax.fill_between(dates, highs, lows, facecolor='blue', alpha=0.1)
--snip--
```

El argumento `alpha` controla la transparencia de un color ❶. Un valor de `alpha` de 0 es completamente transparente, y un valor de 1 (el valor por defecto) es completamente opaco. Ajustando `alpha` a 0,5, hacemos que las líneas de trazado rojas y azules parezcan más claras.

The `alpha` argument controls a color's transparency ❶. An `alpha` value of 0 is completely transparent, and a value of 1 (the default) is completely opaque. By setting `alpha` to 0.5, we make the red and blue plot lines appear lighter.

Pasamos a `fill_between()` la lista `dates` para los `x`-valores y luego las dos `y`-series de valores `highs` y `lows` ❷. El argumento `facecolor` determina el color de la región sombreada; le damos un valor `alpha` bajo de 0,1 para que la región rellena conecte las dos series de datos sin distraer de la información que representan. La Figura 16-5 muestra el gráfico con la región sombreada entre los máximos y los mínimos.

We pass `fill_between()` the list `dates` for the `x`-values and then the two `y`-value series `highs` and `lows` ❷. The `facecolor` argument determines the color of the shaded region; we give it a low `alpha` value of 0.1 so the filled region connects the two data series without distracting from the information they represent. Figure 16-5 shows the plot with the shaded region between the highs and lows.



DESCRIPCIÓN DE LA
IMAGEN (En español):

Temperaturas diarias
máximas y mínimas,
2021. Temperatura
(°F).

Figura 16-5 : La región entre las dos series de datos está sombreada.

Figure 16-5 : The region between the two datasets is shaded.

El sombreado ayuda a hacer inmediatamente evidente el rango entre los dos conjuntos de datos.

The shading helps make the range between the two datasets immediately apparent.

Comprobación de errores (*Error Checking*)

Deberíamos poder ejecutar el código *sitka_highs_lows.py* con los datos de cualquier lugar. Pero algunas estaciones meteorológicas recogen datos distintos de otras, y algunas funcionan mal ocasionalmente y no recogen algunos de los datos que se supone que deben recoger. La falta de datos puede dar lugar a excepciones que bloqueen nuestros programas, a menos que las manejemos adecuadamente.

We should be able to run the *sitka_highs_lows.py* code using data for any location. But some weather stations collect different data than others, and some occasionally malfunction and fail to collect some of the data they're supposed to. Missing data can result in exceptions that crash our programs, unless we handle them properly.

Por ejemplo, veamos qué ocurre cuando intentamos generar un gráfico de temperatura del Valle de la Muerte, California. Copia el archivo *death_valley_2021_simple.csv* en la carpeta donde guardes los datos para los programas de este capítulo.

For example, let's see what happens when we attempt to generate a temperature plot for Death Valley, California. Copy the file *death_valley_2021_simple.csv* to the folder where you're storing the data for this chapter's programs.

Primero, vamos a ejecutar el código para ver las cabeceras que se incluyen en este archivo de datos:

First, let's run the code to see the headers that are included in this data file:

valle_de_la_muerte_altas_bajas.py

death_valley_highs_lows.py

```
from pathlib import Path
import csv

path = Path('weather_data/death_valley_2021_simple.csv')
lines = path.read_text().splitlines()

reader = csv.reader(lines)
header_row = next(reader)

for index, column_header in enumerate(header_row):
    print(index, column_header)
```

Éste es el resultado:

Here's the output:

```
0 STATION
1 NAME
2 DATE
3 TMAX
4 TMIN
5 TOBS
```

La fecha está en la misma posición, en el índice 2. Pero las temperaturas alta y baja están en los índices 3 y 4, por lo que tendremos que cambiar los índices en nuestro código para reflejar estas nuevas posiciones. En lugar de incluir una lectura de la temperatura media del día, esta estación incluye TOBS , una lectura para una hora de observación concreta.

The date is in the same position, at index 2. But the high and low temperatures are at indexes 3 and 4, so we'll need to change the indexes in our code to reflect these new positions. Instead of including an average temperature reading for the day, this station includes TOBS , a reading for a specific observation time.

Cambia *sitka_highs_lows.py* para generar un gráfico para el Valle de la Muerte utilizando los índices que acabamos de anotar, y observa qué ocurre:

Change *sitka_highs_lows.py* to generate a graph for Death Valley using the indexes we just noted, and see what happens:

valle_de_la_muerte_altas_bajas.py

death_valley_highs_lows.py

```
--snip--
path = Path('weather_data/death_valley_2021_simple.csv')
lines = path.read_text().splitlines()
    --snip--
# Extract dates, and high and low temperatures.
dates, highs, lows = [], [], []
for row in reader:
    current_date = datetime.strptime(row[2], '%Y-%m-%d')
    high = int(row[3])
    low = int(row[4])
    dates.append(current_date)
--snip--
```

Actualizamos el programa para que lea del archivo de datos del Valle de la Muerte, y cambiamos los índices para que correspondan a las posiciones TMAX y TMIN de este archivo.

We update the program to read from the Death Valley data file, and we change the indexes to correspond to this file's TMAX and TMIN positions.

Cuando ejecutamos el programa, obtenemos un error:

When we run the program, we get an error:

Traceback (most recent call last):

File "death_valley_highs_lows.py", line 17, in <module>

high = int(row[3])

❶ ValueError: invalid literal for int() with base 10: "

El rastreo nos dice que Python no puede procesar la temperatura alta de una de las fechas porque no puede convertir una cadena vacía (") en un entero ❶ . En lugar de buscar entre los datos para averiguar qué lectura falta, trataremos directamente los casos de datos que faltan.

The traceback tells us that Python can't process the high temperature for one of the dates because it can't turn an empty string (") into an integer ❶ . Rather than looking through the data to find out which reading is missing, we'll just handle cases of missing data directly.

Ejecutaremos código de comprobación de errores cuando se estén leyendo los valores del archivo CSV para gestionar las excepciones que puedan surgir. He aquí cómo hacerlo:

We'll run error-checking code when the values are being read from the CSV file to handle exceptions that might arise. Here's how to do this:

valle_de_la_muerte_altos_bajos.py

death_valley_highs_lows.py

--snip--

for row in reader:

current_date = datetime.strptime(row[2], '%Y-%m-%d')

❶ try:

high = int(row[3])

low = int(row[4])

except ValueError:

❷ print(f"Missing data for {current_date}")

❸ else:

dates.append(current_date)

highs.append(high)

```

        lows.append(low)

# Plot the high and low temperatures.
--snip--

# Format plot.
❹ title = "Daily High and Low Temperatures, 2021\nDeath Valley, CA"
ax.set_title(title, fontsize=20)
ax.set_xlabel("", fontsize=16)
--snip--

```

Cada vez que examinamos una fila, intentamos extraer la fecha y la temperatura alta y baja ❶ . Si falta algún dato, Python lanzará un mensaje `ValueError` y lo gestionaremos imprimiendo un mensaje de error que incluya la fecha del dato que falta ❷ . Tras imprimir el error, el bucle continuará procesando la siguiente fila. Si todos los datos de una fecha se recuperan sin error, se ejecutará el bloque `else` y los datos se añadirán a las listas correspondientes ❸ . Como estamos trazando la información de una nueva ubicación, actualizamos el título para incluir la ubicación en el gráfico, y utilizamos un tamaño de letra más pequeño para acomodar el título más largo ❹ .

Each time we examine a row, we try to extract the date and the high and low temperature ❶ . If any data is missing, Python will raise a `ValueError` and we handle it by printing an error message that includes the date of the missing data ❷ . After printing the error, the loop will continue processing the next row. If all data for a date is retrieved without error, the `else` block will run and the data will be appended to the appropriate lists ❸ . Because we're plotting information for a new location, we update the title to include the location on the plot, and we use a smaller font size to accommodate the longer title ❹ .

Cuando ejecutes ahora *death_valley_highs_lows.py* , verás que sólo faltaban datos de una fecha:

When you run *death_valley_highs_lows.py* now, you'll see that only one date had missing data:

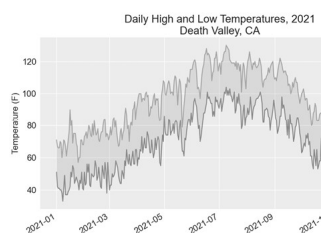
Missing data for 2021-05-04 00:00:00

Como el error se gestiona adecuadamente, nuestro código puede generar un gráfico que omite los datos que faltan. [La Figura 16-6](#) muestra el gráfico resultante.

Because the error is handled appropriately, our code is able to generate a plot, which skips over the missing data. [Figure 16-6](#) shows the resulting plot.

Comparando este gráfico con el de Sitka, podemos ver que el Valle de la Muerte es más cálido en general que el sureste de Alaska, como esperábamos. Además, el rango de temperaturas de cada día es mayor en el desierto. La altura de la región sombreada lo pone de manifiesto.

Comparing this graph to the Sitka graph, we can see that Death Valley is warmer overall than southeast Alaska, as we expect. Also, the range of temperatures each day is greater in the desert. The height of the shaded region makes this clear.



DESCRIPCIÓN DE LA
IMAGEN (En español):

Temperaturas
máximas y mínimas
diarias, 2021 Valle de
la Muerte, CA
Temperatura (°F)
2021-01 2021-03
2021-05 2021-07
2021-09 2021-11
2022-01

Figura 16-6 : Temperaturas máximas y mínimas diarias del Valle de la Muerte

Figure 16-6 : Daily high and low temperatures for Death Valley

Muchos conjuntos de datos con los que trabajes tendrán datos ausentes, mal formateados o incorrectos. Puedes utilizar las herramientas que aprendiste en la primera mitad de este libro para manejar estas situaciones. Aquí utilizamos un bloque `try - except - else` para gestionar los datos que faltan. A veces utilizarás `continue` para saltarte algunos datos, o utilizarás `remove()` o `del` para eliminar algunos datos después de haberlos extraído. Utiliza cualquier método que funcione, siempre que el resultado sea una visualización significativa y precisa.

Many datasets you work with will have missing, improperly formatted, or incorrect data. You can use the tools you learned in the first half of this book to handle these situations.

Here we used a `try - except - else` block to handle missing data. Sometimes you'll use `continue` to skip over some data, or use `remove()` or `del` to eliminate some data after it's been extracted. Use any approach that works, as long as the result is a meaningful, accurate visualization.

Descargar tus propios datos (*Downloading Your Own Data*)

Para descargar tus propios datos meteorológicos, sigue estos pasos:

To download your own weather data, follow these steps:

Listado en español:

1. Visita el sitio de Datos Climáticos en Línea de la NOAA en <https://www.ncdc.noaa.gov/cdo-web> . En la sección Descubre los datos por, haz clic en **Herramienta de búsqueda** . En la casilla Seleccionar un conjunto de datos, elige **Resúmenes diarios** .
2. Selecciona un intervalo de fechas y, en la sección Buscar por, elige **Códigos** postales. Introduce el código postal que te interesa y haz clic en **Buscar** .
3. En la página siguiente, verás un mapa y alguna información sobre la zona en la que te estás centrando. Debajo del nombre de la ubicación, haz clic en **Ver detalles completos** , o haz clic en el mapa y luego en **Detalles completos** .
4. Desplázate hacia abajo y haz clic en **Lista de estaciones** para ver las estaciones meteorológicas disponibles en esta zona. Haz clic en el nombre de una de las estaciones y luego haz clic en **Añadir a la Cesta** . Estos datos son gratuitos, aunque el sitio utiliza un icono de carrito de la compra. En la esquina superior derecha, haz clic en el carrito.
5. En Seleccionar el formato de salida, elige **Personalizado GHCN-Daily CSV** . Asegúrate de que el intervalo de fechas es correcto y haz clic en **Continuar** .
6. En la página siguiente, puedes seleccionar los tipos de datos que desees. Puedes descargar un tipo de datos (por ejemplo, centrarte en la temperatura del aire) o puedes descargar todos los datos disponibles de esta estación. Haz tu elección y haz clic en **Continuar** .
7. En la última página, verás un resumen de tu pedido. Introduce tu dirección de correo electrónico y haz clic en **Enviar pedido** . Recibirás una confirmación de que se ha recibido tu pedido y, en unos minutos, deberías recibir otro correo electrónico con un enlace para descargar tus datos.

Listing in English:

1. Visit the NOAA Climate Data Online site at <https://www.ncdc.noaa.gov/cdo-web> . In the Discover Data By section, click **Search Tool** . In the Select a Dataset box, choose **Daily Summaries** .
2. Select a date range, and in the Search For section, choose **ZIP Codes** . Enter the ZIP code you're interested in and click **Search** .
3. On the next page, you'll see a map and some information about the area you're focusing on. Below the location name, click **View Full Details** , or click the map and then click **Full Details** .
4. Scroll down and click **Station List** to see the weather stations that are available in this area. Click one of the station names and then click **Add to Cart** . This data is free, even though the site uses a shopping cart icon. In the upper-right corner, click the cart.
5. In Select the Output Format, choose **Custom GHCN-Daily CSV** . Make sure the date range is correct and click **Continue** .
6. On the next page, you can select the kinds of data you want. You can download one kind of data (for example, focusing on air temperature) or you can download all the data available from this station. Make your choices and then click **Continue** .
7. On the last page, you'll see a summary of your order. Enter your email address and click **Submit Order** . You'll receive a confirmation that your order was received, and in a few minutes, you should receive another email with a link to download your data.

Los datos que descargues deben tener la misma estructura que los datos con los que hemos trabajado en esta sección. Puede que tengan cabeceras diferentes a las que has visto en esta sección, pero si sigues los mismos pasos que hemos utilizado aquí, deberías poder generar visualizaciones de los datos que te interesan.

The data you download should be structured just like the data we worked with in this section. It might have different headers than those you saw in this section, but if you follow the same steps we used here, you should be able to generate visualizations of the data you're interested in.

Pruébalo tú mismo (Try It Yourself)

16-1. Precipitaciones en Sitka: Sitka está situada en un bosque templado lluvioso, por lo que recibe una buena cantidad de precipitaciones. En el archivo de datos *sitka_weather_2021_full.csv* hay una cabecera llamada **PRCP** , que representa las cantidades de lluvia diarias. Haz una visualización centrada en los datos de esta columna. Puedes repetir el ejercicio para el Valle de la Muerte si tienes curiosidad por saber lo poco que llueve en un desierto.

16-1. Sitka Rainfall: Sitka is located in a temperate rainforest, so it gets a fair amount of rainfall. In the data file *sitka_weather_2021_full.csv* is a header called `PRCP`, which represents daily rainfall amounts. Make a visualization focusing on the data in this column. You can repeat the exercise for Death Valley if you're curious how little rainfall occurs in a desert.

16-2. Comparación Sitka-Valle de la Muerte: Las escalas de temperatura de los gráficos de Sitka y del Valle de la Muerte reflejan los diferentes rangos de datos. Para comparar con precisión el rango de temperaturas de Sitka con el del Valle de la Muerte, necesitas escalas idénticas en el eje y . Cambia los ajustes del eje y en uno o ambos gráficos de las Figuras 16-5 y 16-6. Luego haz una comparación directa entre los rangos de temperatura de Sitka y del Valle de la Muerte (o de cualquier otro lugar que quieras comparar).

16-2. Sitka–Death Valley Comparison: The temperature scales on the Sitka and Death Valley graphs reflect the different data ranges. To accurately compare the temperature range in Sitka to that of Death Valley, you need identical scales on the y -axis. Change the settings for the y -axis on one or both of the charts in Figures 16-5 and 16-6. Then make a direct comparison between temperature ranges in Sitka and Death Valley (or any two places you want to compare).

16-3. San Francisco: ¿Se parecen más las temperaturas de San Francisco a las de Sitka o a las del Valle de la Muerte? Descarga algunos datos de San Francisco, y genera un gráfico de temperaturas altas-bajas de San Francisco para hacer una comparación.

16-3. San Francisco: Are temperatures in San Francisco more like temperatures in Sitka or temperatures in Death Valley? Download some data for San Francisco, and generate a high-low temperature plot for San Francisco to make a comparison.

16-4. Índices automáticos: En esta sección, hemos codificado los índices correspondientes a las columnas `TMIN` y `TMAX`. Utiliza la fila de cabecera para determinar los índices de estos valores, de forma que tu programa pueda funcionar para Sitka o el Valle de la Muerte. Utiliza también el nombre de la estación para generar automáticamente un título apropiado para tu gráfico.

16-4. Automatic Indexes: In this section, we hardcoded the indexes corresponding to the TMIN and TMAX columns. Use the header row to determine the indexes for these values, so your program can work for Sitka or Death Valley. Use the station name to automatically generate an appropriate title for your graph as well.

16-5. Explora: Genera algunas visualizaciones más que examinen cualquier otro aspecto meteorológico que te interese para cualquier ubicación sobre la que sientas curiosidad.

16-5. Explore: Generate a few more visualizations that examine any other weather aspect you're interested in for any locations you're curious about.

Cartografía de Conjuntos de Datos Globales: Formato GeoJSON (*Mapping Global Datasets: GeoJSON Format*)

En esta sección, descargarás un conjunto de datos que representa todos los terremotos ocurridos en el mundo durante el mes anterior. Luego harás un mapa que muestre la ubicación de estos terremotos y la importancia de cada uno de ellos. Como los datos están almacenados en formato GeoJSON, trabajaremos con ellos utilizando el módulo `json`. Utilizando el gráfico `scatter_geo()` de Plotly, crearás visualizaciones que muestren claramente la distribución global de los terremotos.

In this section, you'll download a dataset representing all the earthquakes that have occurred in the world during the previous month. Then you'll make a map showing the location of these earthquakes and how significant each one was. Because the data is stored in the GeoJSON format, we'll work with it using the `json` module. Using Plotly's `scatter_geo()` plot, you'll create visualizations that clearly show the global distribution of earthquakes.

Descargar datos de terremotos (*Downloading Earthquake Data*)

Crea una carpeta llamada `eq_data` dentro de la carpeta donde guardas los programas de este capítulo. Copia el archivo `eq_1_day_m1.geojson` en esta nueva carpeta. Los terremotos se clasifican por su magnitud en la escala de Richter. Este archivo incluye los datos de todos los terremotos de magnitud M1 o superior ocurridos en las últimas 24 horas (en el momento de escribir esto). Estos datos proceden de una de las fuentes de datos sobre terremotos del Servicio Geológico de Estados Unidos, en

<https://earthquake.usgs.gov/earthquakes/feed> .

Make a folder called *eq_data* inside the folder where you're saving this chapter's programs. Copy the file *eq_1_day_m1.geojson* into this new folder. Earthquakes are categorized by their magnitude on the Richter scale. This file includes data for all earthquakes with a magnitude M1 or greater that took place in the last 24 hours (at the time of this writing). This data comes from one of the United States Geological Survey's earthquake data feeds, at <https://earthquake.usgs.gov/earthquakes/feed> .

Examinar los datos GeoJSON (Examining GeoJSON Data)

Cuando abras *eq_1_day_m1.geojson* , verás que es muy denso y difícil de leer:

When you open *eq_1_day_m1.geojson* , you'll see that it's very dense and hard to read:

```
{"type":"FeatureCollection","metadata":{"generated":1649052296000,...  
{"type":"Feature","properties":{"mag":1.6,"place":"63 km SE of Ped...  
{"type":"Feature","properties":{"mag":2.2,"place":"27 km SSE of Ca...  
{"type":"Feature","properties":{"mag":3.7,"place":"102 km SSE of S...  
{"type":"Feature","properties":{"mag":2.92000008,"place":"49 km SE...  
{"type":"Feature","properties":{"mag":1.4,"place":"44 km NE of Sus...  
--snip--
```

Este archivo está formateado más para máquinas que para humanos. Pero podemos ver que el archivo contiene algunos diccionarios, así como información que nos interesa, como las magnitudes y ubicaciones de los terremotos.

This file is formatted more for machines than humans. But we can see that the file contains some dictionaries, as well as information that we're interested in, such as earthquake magnitudes and locations.

El módulo `json` proporciona diversas herramientas para explorar y trabajar con datos JSON. Algunas de estas herramientas nos ayudarán a reformatear el archivo para que podamos ver los datos en bruto más fácilmente antes de trabajar con ellos mediante programación.

The `json` module provides a variety of tools for exploring and working with JSON data. Some of these tools will help us reformat the file so we can look at the raw data more easily before we work with it programmatically.

Empecemos por cargar los datos y mostrarlos en un formato más fácil de leer. Se trata de un archivo de datos largo, así que en lugar de imprimirlo, reescribiremos los datos en un nuevo archivo. Así podremos abrir ese archivo y desplazarnos hacia adelante y hacia atrás por los datos más fácilmente:

Let's start by loading the data and displaying it in a format that's easier to read. This is a long data file, so instead of printing it, we'll rewrite the data to a new file. Then we can open that file and scroll back and forth through the data more easily:

eq_explorar_datos.py

eq_explore_data.py

```
from pathlib import Path
import json

# Read data as a string and convert to a Python object.
path = Path('eq_data/eq_data_1_day_m1.geojson')
contents = path.read_text()
❶ all_eq_data = json.loads(contents)

# Create a more readable version of the data file.
❷ path = Path('eq_data/readable_eq_data.geojson')
❸ readable_contents = json.dumps(all_eq_data, indent=4)
path.write_text(readable_contents)
```

Leemos el archivo de datos como una cadena, y utilizamos `json.loads()` para convertir la representación de cadena del archivo en un objeto Python ❶. Se trata del mismo enfoque que utilizamos en el Capítulo 10. En este caso, todo el conjunto de datos se convierte en un único diccionario, que asignamos a `all_eq_data`. A continuación, definimos un nuevo `path` donde podemos escribir estos mismos datos en un formato más legible ❷. La función `json.dumps()` que viste en el Capítulo 10 puede tomar un argumento opcional `indent` ❸, que le indica cuánto sangrar los elementos anidados en la estructura de datos.

We read the data file as a string, and use `json.loads()` to convert the string representation of the file to a Python object ❶. This is the same approach we used in Chapter 10. In this case, the entire dataset is converted to a single dictionary, which we assign to `all_eq_data`. We then define a new `path` where we can write this same data in a more readable format ❷. The `json.dumps()` function that you saw in Chapter 10 can take an optional `indent` argument ❸, which tells it how much to indent nested elements in the data structure.

Cuando busques en tu directorio `eq_data` y abras el archivo `readable_eq_data.json`, esto es lo primero que verás:

When you look in your `eq_data` directory and open the file `readable_eq_data.json`, here's the first part of what you'll see:

datos_leibles_eq.json

readable_eq_data.json

```
{
  "type": "FeatureCollection",
  ❶ "metadata": {
    "generated": 1649052296000,
    "url": "https://earthquake.usgs.gov/earthquakes/.../1.0_day.geojson",
    "title": "USGS Magnitude 1.0+ Earthquakes, Past Day",
    "status": 200,
    "api": "1.10.3",
    "count": 160
  },
  ❷ "features": [
    --snip--
```

La primera parte del archivo incluye una sección con la clave `"metadata"` ❶. Esto nos indica cuándo se generó el archivo de datos y dónde podemos encontrar los datos en línea. También nos da un título legible por humanos y el número de terremotos incluidos en este archivo. En este periodo de 24 horas, se registraron 160 terremotos.

The first part of the file includes a section with the key "metadata" ❶ . This tells us when the data file was generated and where we can find the data online. It also gives us a human-readable title and the number of earthquakes included in this file. In this 24-hour period, 160 earthquakes were recorded.

Este archivo GeoJSON tiene una estructura que resulta útil para los datos basados en la ubicación. La información se almacena en una lista asociada a la clave "features" ❷ . Como este archivo contiene datos sobre terremotos, los datos están en forma de lista, donde cada elemento de la lista corresponde a un único terremoto. Esta estructura puede parecer confusa, pero es bastante potente. Permite a los geólogos almacenar toda la información que necesiten en un diccionario sobre cada terremoto, y luego meter todos esos diccionarios en una gran lista.

This GeoJSON file has a structure that's helpful for location-based data. The information is stored in a list associated with the key "features" ❷ . Because this file contains earthquake data, the data is in list form where every item in the list corresponds to a single earthquake. This structure might look confusing, but it's quite powerful. It allows geologists to store as much information as they need to in a dictionary about each earthquake, and then stuff all those dictionaries into one big list.

Veamos un diccionario que representa un único terremoto:

Let's look at a dictionary representing a single earthquake:

datos_leibles.json

readable_eq_data.json

```
--snip--
{
  "type": "Feature",
  ❶ "properties": {
```



```

        "mag": 1.6,
        --snip--
    ❷      "title": "M 1.6 - 27 km NNW of Susitna, Alaska"
    },
    ❸      "geometry": {
        "type": "Point",
        "coordinates": [
    ❹          -150.7585,
    ❺          61.7591,
            56.3
        ]
    },
    "id": "ak0224bj1jx"
},

```

La clave "properties" contiene mucha información sobre cada terremoto ❶ . Nos interesa principalmente la magnitud de cada terremoto, asociada a la clave "mag" . También nos interesa la "title" de cada suceso, que proporciona un buen resumen de su magnitud y ubicación ❷ .

The key "properties" contains a lot of information about each earthquake ❶ . We're mainly interested in the magnitude of each earthquake, associated with the key "mag" . We're also interested in the "title" of each event, which provides a nice summary of its magnitude and location ❷ .

La clave "geometry" nos ayuda a comprender dónde se produjo el terremoto ❸ . Necesitaremos esta información para cartografiar cada suceso. Podemos encontrar la longitud ❹ y la latitud ❺ de cada terremoto en una lista asociada a la clave "coordinates" .

The key "geometry" helps us understand where the earthquake occurred ❸ . We'll need this information to map each event. We can find the longitude ❹ and the latitude ❺ for each earthquake in a list associated with the key "coordinates" .

Este archivo contiene mucho más anidamiento del que utilizaríamos en el código que escribimos, así que si parece confuso, no te preocupes: Python se encargará de la mayor parte de la complejidad. Sólo trabajaremos con uno o dos niveles de anidamiento cada vez. Empezaremos sacando un diccionario para cada terremoto que se haya registrado en el

periodo de 24 horas.

This file contains way more nesting than we'd use in the code we write, so if it looks confusing, don't worry: Python will handle most of the complexity. We'll only be working with one or two nesting levels at a time. We'll start by pulling out a dictionary for each earthquake that was recorded in the 24-hour time period.

Nota (Note)

Cuando hablamos de lugares, a menudo decimos primero la latitud del lugar, seguida de su longitud. Esta convención surgió probablemente porque los humanos descubrimos la latitud mucho antes de que desarrolláramos el concepto de longitud. Sin embargo, muchos marcos geoespaciales indican primero la longitud y luego la latitud, porque esto se corresponde con la convención (x, y) que utilizamos en las representaciones matemáticas. El formato GeoJSON sigue la convención (longitud, latitud). Si utilizas un marco diferente, es importante que aprendas qué convención sigue ese marco.

When we talk about locations, we often say the location's latitude first, followed by its longitude. This convention probably arose because humans discovered latitude long before we developed the concept of longitude. However, many geospatial frameworks list the longitude first and then the latitude, because this corresponds to the (x, y) convention we use in mathematical representations. The GeoJSON format follows the (longitude, latitude) convention. If you use a different framework, it's important to learn what convention that framework follows.

Hacer una lista de todos los terremotos (Making a List of All Earthquakes)

En primer lugar, haremos una lista que contenga toda la información sobre cada terremoto ocurrido.

First, we'll make a list that contains all the information about every earthquake that occurred.

eq_explorar_datos.py

eq_explore_data.py

```
from pathlib import Path
import json

# Read data as a string and convert to a Python object.
path = Path('eq_data/eq_data_1_day_m1.geojson')
contents = path.read_text()
all_eq_data = json.loads(contents)

# Examine all earthquakes in the dataset.
all_eq_dicts = all_eq_data['features']
print(len(all_eq_dicts))
```

Tomamos los datos asociados a la clave 'features' en el diccionario `all_eq_data`, y los asignamos a `all_eq_dicts`. Sabemos que este archivo contiene registros de 160 terremotos, y la salida verifica que hemos capturado todos los terremotos del archivo:

We take the data associated with the key 'features' in the `all_eq_data` dictionary, and assign it to `all_eq_dicts`. We know this file contains records of 160 earthquakes, and the output verifies that we've captured all the earthquakes in the file:

160

Fíjate en lo corto que es este código. El archivo *readable_eq_data.json*, pulcramente formateado, tiene más de 6.000 líneas. Pero en sólo unas líneas, podemos leer todos esos datos y almacenarlos en una lista de Python. A continuación, sacaremos las magnitudes de cada terremoto.

Notice how short this code is. The neatly formatted file *readable_eq_data.json* has over 6,000 lines. But in just a few lines, we can read through all that data and store it in a Python list. Next, we'll pull the magnitudes from each earthquake.

Extraer magnitudes (*Extracting Magnitudes*)

Podemos recorrer la lista que contiene los datos de cada terremoto y extraer la información que queramos. Extraigamos la magnitud de cada terremoto:

We can loop through the list containing data about each earthquake, and extract any information we want. Let's pull out the magnitude of each earthquake:

eq_explorar_datos.py

eq_explore_data.py

```
--snip--
all_eq_dicts = all_eq_data['features']

❶ mags = []
for eq_dict in all_eq_dicts:
    ❷ mag = eq_dict['properties']['mag']
    mags.append(mag)

print(mags[:10])
```

Creamos una lista vacía para almacenar las magnitudes, y luego hacemos un bucle a través de la lista `all_eq_dicts` ❶. Dentro de este bucle, cada terremoto está representado por el diccionario `eq_dict`. La magnitud de cada terremoto se almacena en la sección `'properties'` de este diccionario, bajo la clave `'mag'` ❷. Almacenamos cada magnitud en la variable `mag` y luego la añadimos a la lista `mags`.

We make an empty list to store the magnitudes, and then loop through the list `all_eq_dicts` ❶. Inside this loop, each earthquake is represented by the dictionary `eq_dict`. Each earthquake's magnitude is stored in the `'properties'` section of this dictionary, under the key `'mag'` ❷. We store each magnitude in the variable `mag` and then append it to the list `mags`.

Imprimimos las primeras magnitudes 10, para saber si estamos obteniendo los datos correctos:

We print the first 10 magnitudes, so we can see whether we're getting the correct data:

```
[1.6, 1.6, 2.2, 3.7, 2.92000008, 1.4, 4.6, 4.5, 1.9, 1.8]
```

A continuación, sacaremos los datos de localización de cada terremoto, y así podremos hacer un mapa de los terremotos.

Next, we'll pull the location data for each earthquake, and then we can make a map of the earthquakes.

Extraer los datos de localización *(Extracting Location Data)*

Los datos de localización de cada terremoto se almacenan bajo la clave "geometry" . Dentro del diccionario de geometría hay una clave "coordinates" , y los dos primeros valores de esta lista son la longitud y la latitud. He aquí cómo extraeremos estos datos:

The location data for each earthquake is stored under the key "geometry" . Inside the geometry dictionary is a "coordinates" key, and the first two values in this list are the longitude and latitude. Here's how we'll pull this data:

eq_explorar_datos.py

eq_explore_data.py

```
--snip--
all_eq_dicts = all_eq_data['features']

mags, lons, lats = [], [], []
for eq_dict in all_eq_dicts:
    mag = eq_dict['properties']['mag']
    ❶ lon = eq_dict['geometry']['coordinates'][0]
    lat = eq_dict['geometry']['coordinates'][1]
```

```
mags.append(mag)
lons.append(lon)
lats.append(lat)

print(mags[:10])
print(lons[:5])
print(lats[:5])
```

Hacemos listas vacías para las longitudes y latitudes. El código `eq_dict['geometry']` accede al diccionario que representa el elemento de geometría del terremoto ❶. La segunda clave, 'coordinates', extrae la lista de valores asociada a 'coordinates'. Por último, el índice 0 pide el primer valor de la lista de coordenadas, que corresponde a la longitud de un terremoto.

We make empty lists for the longitudes and latitudes. The code `eq_dict['geometry']` accesses the dictionary representing the geometry element of the earthquake ❶. The second key, 'coordinates', pulls the list of values associated with 'coordinates'. Finally, the 0 index asks for the first value in the list of coordinates, which corresponds to an earthquake's longitude.

Cuando imprimimos las primeras longitudes y latitudes de 5, la salida muestra que estamos extrayendo los datos correctos:

When we print the first 5 longitudes and latitudes, the output shows that we're pulling the correct data:

```
[1.6, 1.6, 2.2, 3.7, 2.92000008, 1.4, 4.6, 4.5, 1.9, 1.8]
[-150.7585, -153.4716, -148.7531, -159.6267, -155.248336791992]
[61.7591, 59.3152, 63.1633, 54.5612, 18.7551670074463]
```

Con estos datos, podemos pasar a cartografiar cada terremoto.

With this data, we can move on to mapping each earthquake.

Construir un mapa del mundo (*Building a World Map*)

Con la información que hemos obtenido hasta ahora, podemos construir un sencillo mapamundi. Aunque todavía no tendrá un aspecto presentable, queremos asegurarnos de que la información se muestra correctamente antes de centrarnos en cuestiones de estilo y presentación. Éste es el mapa inicial:

Using the information we've pulled so far, we can build a simple world map. Although it won't look presentable yet, we want to make sure the information is displayed correctly before focusing on style and presentation issues. Here's the initial map:

eq_mapa_del_mundo.py

eq_world_map.py

```
from pathlib import Path
import json

import plotly.express as px

--snip--
for eq_dict in all_eq_dicts:
    --snip--

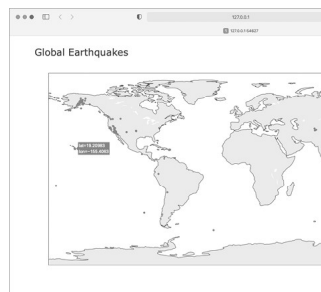
title = 'Global Earthquakes'
❶ fig = px.scatter_geo(lat=lats, lon=lons, title=title)
fig.show()
```

Importamos `plotly.express` con el alias `px`, igual que hicimos en el Capítulo 15. La función `scatter_geo()` ❶ te permite superponer un gráfico de dispersión de datos geográficos sobre un mapa. En el uso más sencillo de este tipo de gráfico, sólo necesitas proporcionar una lista de latitudes y otra de longitudes. Pasamos la lista `lats` al argumento `lat`, y `lons` al argumento `lon`.

We import `plotly.express` with the alias `px`, just as we did in Chapter 15. The `scatter_geo()` function ❶ allows you to overlay a scatterplot of geographic data on a map. In the simplest use of this chart type, you only need to provide a list of latitudes and a list of longitudes. We pass the list `lats` to the `lat` argument, and `lons` to the `lon` argument.

Cuando ejecutes este archivo, deberías ver un mapa parecido al de la [Figura 16-7](#) . Esto demuestra una vez más la potencia de la biblioteca Plotly Express; en sólo tres líneas de código, tenemos un mapa de la actividad sísmica mundial.

When you run this file, you should see a map that looks like the one in [Figure 16-7](#) . This again shows the power of the Plotly Express library; in just three lines of code, we have a map of global earthquake activity.



DESCRIPCIÓN DE LA IMAGEN (En español):

Terremotos Globales
La imagen muestra un mapa del mundo con puntos que indican la ubicación de terremotos. En el océano Pacífico, cerca de la costa oeste de América del Norte, hay detalles adicionales que destacan un área con las siguientes coordenadas:
"lat=19.246093" y
"lon=-105.6094". Este mapa parece estar enmarcado dentro de una ventana de navegador con la dirección "127.0.0.1" en la barra de búsqueda y el número de puerto "7867".

Figura 16-7 : Un sencillo mapa que muestra dónde se produjeron todos los terremotos de las últimas 24 horas

Figure 16-7 : A simple map showing where all the earthquakes in the last 24 hours occurred

Ahora que sabemos que la información de nuestro conjunto de datos se traza correctamente, podemos hacer algunos cambios para que el mapa tenga más sentido y sea más fácil de leer.

Now that we know the information in our dataset is being plotted correctly, we can make a few changes to make the map more meaningful and easier to read.

Representación de magnitudes (*Representing Magnitudes*)

Un mapa de actividad sísmica debe mostrar la magnitud de cada terremoto. También podemos incluir más datos, ahora que sabemos que los datos se trazan correctamente.

A map of earthquake activity should show the magnitude of each earthquake. We can also include more data, now that we know the data is being plotted correctly.

```
--snip--
# Read data as a string and convert to a Python object.
path = Path('eq_data/eq_data_30_day_m1.geojson')
contents = path.read_text()
--snip--

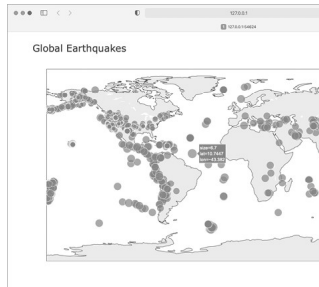
title = 'Global Earthquakes'
fig = px.scatter_geo(lat=lats, lon=lons, size=mags, title=title)
fig.show()
```

Cargamos el archivo *eq_data_30_day_m1.geojson* , para incluir 30 días completos de actividad sísmica. También utilizamos el argumento tamaño en la llamada a *px.scatter_geo()* , que especifica cómo se dimensionarán los puntos del mapa. Pasamos la lista *mags* a *size* , para que los terremotos de mayor magnitud aparezcan como puntos más grandes en el mapa.

We load the file *eq_data_30_day_m1.geojson* , to include a full 30 days' worth of earthquake activity. We also use the size argument in the *px.scatter_geo()* call, which specifies how the points on the map will be sized. We pass the list *mags* to *size* , so earthquakes with a higher magnitude will show up as larger points on the map.

El mapa resultante se muestra en la [Figura 16-8](#) . Los terremotos suelen producirse cerca de los límites de las placas tectónicas, y el periodo más largo de actividad sísmica incluido en este mapa revela las ubicaciones exactas de estos límites.

The resulting map is shown in [Figure 16-8](#) . Earthquakes usually occur near tectonic plate boundaries, and the longer period of earthquake activity included in this map reveals the exact locations of these boundaries.



DESCRIPCIÓN DE LA
IMAGEN (En español):

Terremotos Globales
En el mapa se muestran múltiples puntos que representan terremotos a nivel mundial. Uno de los puntos destacados indica las coordenadas: latitud 0.47, longitud -10.75, y profundidad -34.52.

Figura 16-8 : El mapa muestra ahora la magnitud de todos los terremotos de los últimos 30 días.

Figure 16-8 : The map now shows the magnitude of all earthquakes in the last 30 days.

Este mapa es mejor, pero sigue siendo difícil distinguir qué puntos representan los terremotos más importantes. Podemos mejorarlo aún más utilizando también el color para representar las magnitudes.

This map is better, but it's still difficult to pick out which points represent the most significant earthquakes. We can improve this further by using color to represent magnitudes as well.

Personalizar los colores de los marcadores (Customizing Marker Colors)

Podemos utilizar las escalas de color de Plotly para personalizar el color de cada marcador, según la gravedad del terremoto correspondiente. También utilizaremos una proyección diferente para el mapa base.

We can use Plotly's color scales to customize each marker's color, according to the severity of the corresponding earthquake. We'll also use a different projection for the base map.

eq_mundi_mapa.py

eq_world_map.py

```
--snip--
fig = px.scatter_geo(lat=lats, lon=lons, size=mags, title=title,
①    color=mags,
②    color_continuous_scale='Viridis',
③    labels={'color':'Magnitude'},
④    projection='natural earth',
    )
fig.show()
```

Todos los cambios significativos aquí se producen en la llamada a la función `px.scatter_geo()`. El argumento `color` indica a Plotly qué valores debe utilizar para determinar dónde se sitúa cada marcador en la escala de colores ①. Utilizamos la lista `mags` para determinar el color de cada punto, igual que hicimos con el argumento `size`.

All the significant changes here occur in the `px.scatter_geo()` function call. The `color` argument tells Plotly what values it should use to determine where each marker falls on the color scale ①. We use the `mags` list to determine the color for each point, just as we did with the `size` argument.

El argumento `color_continuous_scale` indica a Plotly qué escala de colores debe utilizar ②. *Viridis* es una escala de colores que va del azul oscuro al amarillo brillante, y funciona bien para este conjunto de datos. Por defecto, la escala de colores de la derecha del mapa está etiquetada como *color*; esto no es representativo de lo que significan realmente los colores. El argumento `labels`, mostrado en el Capítulo 15, toma un diccionario como valor ③. Sólo tenemos que establecer una etiqueta personalizada en este gráfico, asegurándonos de que la escala de colores se etiqueta *Magnitude* en lugar de *color*.

The `color_continuous_scale` argument tells Plotly which color scale to use ②. *Viridis* is a color scale that ranges from dark blue to bright yellow, and it works well for this dataset.

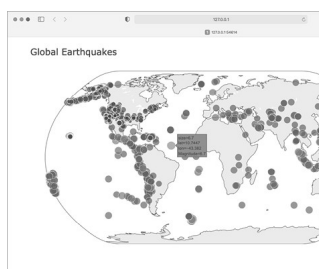
By default, the color scale on the right of the map is labeled *color* ; this is not representative of what the colors actually mean. The `labels` argument, shown in Chapter 15 , takes a dictionary as a value ③ . We only need to set one custom label on this chart, making sure the color scale is labeled *Magnitude* instead of *color* .

Añadimos un argumento más, para modificar el mapa base sobre el que se trazan los terremotos. El argumento `projection` acepta una serie de proyecciones cartográficas comunes ④ . Aquí utilizamos la proyección 'natural earth' , que redondea los extremos del mapa. Fíjate también en la coma al final de este último argumento. Cuando una llamada a una función tiene una larga lista de argumentos que abarca varias líneas como ésta, es práctica común añadir una coma al final para que siempre estés preparado para añadir otro argumento en la línea siguiente.

We add one more argument, to modify the base map over which the earthquakes are plotted. The `projection` argument accepts a number of common map projections ④ . Here we use the 'natural earth' projection, which rounds the ends of the map. Also, note the trailing comma after this last argument. When a function call has a long list of arguments spanning multiple lines like this, it's common practice to add a trailing comma so you're always ready to add another argument on the next line.

Cuando ejecutes ahora el programa, verás un mapa de aspecto mucho más agradable. En la [Figura 16-9](#) , la escala de colores muestra la gravedad de los terremotos individuales; los terremotos más graves destacan como puntos de color amarillo claro, en contraste con muchos puntos más oscuros. También puedes saber qué regiones del mundo tienen una actividad sísmica más significativa.

When you run the program now, you'll see a much nicer-looking map. In [Figure 16-9](#) , the color scale shows the severity of individual earthquakes; the most severe earthquakes stand out as light-yellow points, in contrast to many darker points. You can also tell which regions of the world have more significant earthquake activity.



DESCRIPCIÓN DE LA
IMAGEN (En español):

Terremotos Globales.
El mapa muestra la
distribución mundial
de terremotos
representada por
puntos en diferentes
tonos de gris. Una

leyenda al lado derecho indica la magnitud, numerada de 5 a 9, lo que sugiere que los tonos más oscuros representan terremotos de mayor magnitud. Un cuadro emergente muestra un detalle de un evento específico: "9 de enero de 47, Latitud: -7.62, Longitud: 108.38, Magnitud: 6.7". La región del mapa destaca especialmente los bordes de las placas tectónicas donde se concentran muchos de los puntos.

Figura 16-9 : En 30 días de terremotos, el color y el tamaño se utilizan para representar la magnitud de cada terremoto.

Figure 16-9 : In 30 days' worth of earthquakes, color and size are used to represent the magnitude of each earthquake.

Otras escalas de color (Other Color Scales)

Puedes elegir entre otras escalas de color. Para ver las escalas de color disponibles, introduce las dos líneas siguientes en una sesión de terminal Python:

You can choose from a number of other color scales. To see the available color scales, enter the following two lines in a Python terminal session:

```
>>> import plotly.express as px
>>> px.colors.named_colorscale()
['aggrnyl', 'agsunset', 'blackbody', ..., 'mygbm']
```

No dudes en probar estas escalas de color en el mapa de terremotos, o con cualquier conjunto de datos en el que la variación continua de los colores pueda ayudar a mostrar patrones en los datos.

Feel free to try out these color scales in the earthquake map, or with any dataset where continuously varying colors can help show patterns in the data.

Añadir texto sobreimpresionado (*Adding Hover Text*)

Para terminar este mapa, añadiremos un texto informativo que aparecerá cuando pases el ratón por encima del marcador que representa un terremoto. Además de mostrar la longitud y la latitud, que aparecen por defecto, mostraremos también la magnitud y proporcionaremos una descripción de la ubicación aproximada.

To finish this map, we'll add some informative text that appears when you hover over the marker representing an earthquake. In addition to showing the longitude and latitude, which appear by default, we'll show the magnitude and provide a description of the approximate location as well.

Para realizar este cambio, necesitamos extraer un poco más de datos del archivo:

To make this change, we need to pull a little more data from the file:

eq_mundi_mapa.py

eq_world_map.py

--snip--

```
❶ mags, lons, lats, eq_titles = [], [], [], []
    mag = eq_dict['properties']['mag']
    lon = eq_dict['geometry']['coordinates'][0]
    lat = eq_dict['geometry']['coordinates'][1]
❷ eq_title = eq_dict['properties']['title']
    mags.append(mag)
    lons.append(lon)
    lats.append(lat)
    eq_titles.append(eq_title)
```

```
title = 'Global Earthquakes'
```

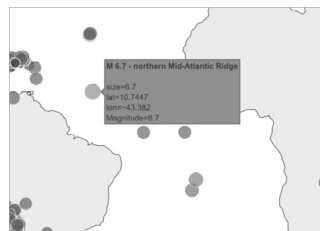
```
fig = px.scatter_geo(lat=lats, lon=lons, size=mags, title=title,
    --snip--
    projection='natural earth',
    ❸    hover_name=eq_titles,
    )
fig.show()
```

Primero hacemos una lista llamada `eq_titles` para almacenar el título de cada terremoto ❶ . La sección `'title'` de los datos contiene un nombre descriptivo de la magnitud y ubicación de cada terremoto, además de su longitud y latitud. Extraemos esta información y la asignamos a la variable `eq_title` ❷ , y luego la añadimos a la lista `eq_titles` .

We first make a list called `eq_titles` to store the title of each earthquake ❶ . The `'title'` section of the data contains a descriptive name of the magnitude and location of each earthquake, in addition to its longitude and latitude. We pull this information and assign it to the variable `eq_title` ❷ , and then append it to the list `eq_titles` .

En la llamada a `px.scatter_geo()` , pasamos `eq_titles` al argumento `hover_name` ❸ . Plotly añadirá ahora la información del título de cada terremoto al texto hover de cada punto. Cuando ejecutes este programa, deberías poder pasar el ratón por encima de cualquier marcador, ver una descripción de dónde tuvo lugar ese terremoto y leer su magnitud exacta. En la [Figura 16-10](#) se muestra un ejemplo de esta información.

In the `px.scatter_geo()` call, we pass `eq_titles` to the `hover_name` argument ❸ . Plotly will now add the information from the title of each earthquake to the hover text on each point. When you run this program, you should be able to hover over any marker, see a description of where that earthquake took place, and read its exact magnitude. An example of this information is shown in [Figure 16-10](#) .



DESCRIPCIÓN DE LA
IMAGEN (En español):

M 6.7 - cresta del norte
del Atlántico medio
tamaño = 6.7 lat =
10.7447 lon = -43.382
Magnitud = 6.7

Figura 16-10 : El texto que aparece al pasar el ratón incluye ahora un resumen de cada terremoto.

Figure 16-10 : The hover text now includes a summary of each earthquake.

¡Es impresionante! En menos de 30 líneas de código, hemos creado un mapa visualmente atractivo y significativo de la actividad sísmica mundial que también ilustra en la estructura geológica del planeta. Plotly ofrece una amplia gama de formas de personalizar el aspecto y el comportamiento de tus visualizaciones. Utilizando las numerosas opciones de Plotly, puedes hacer gráficos y mapas que muestren exactamente lo que quieres que muestren.

This is impressive! In less than 30 lines of code, we've created a visually appealing and meaningful map of global earthquake activity that also illustrates the geological structure of the planet. Plotly offers a wide range of ways you can customize the appearance and behavior of your visualizations. Using Plotly's many options, you can make charts and maps that show exactly what you want them to.

Pruébalo tú mismo (*Try It Yourself*)

16-6. Refactorización: El bucle que extrae datos de `all_eq_dicts` utiliza variables para la magnitud, la longitud, la latitud y el título de cada terremoto antes de añadir estos valores a sus listas correspondientes. Este enfoque se eligió para aclarar cómo extraer datos de un archivo GeoJSON, pero no es necesario en tu código. En lugar de utilizar estas variables temporales, extrae cada valor de `eq_dict` y añádelo a la lista correspondiente en una sola línea. Haciendo esto acortarás el cuerpo de este bucle a sólo cuatro líneas.

16-6. Refactoring: The loop that pulls data from `all_eq_dicts` uses variables for the magnitude, longitude, latitude, and title of each earthquake before appending these values to their appropriate lists. This approach was chosen for clarity in how to pull data from a GeoJSON file, but it's not necessary in your code. Instead of using these temporary variables, pull each value from `eq_dict` and append it to the appropriate list in one line. Doing so should shorten the body of this loop to just four lines.

16-7. Título automatizado: En esta sección, hemos utilizado el título genérico *Global Earthquakes*. En su lugar, puedes utilizar el título del conjunto de datos en la parte de metadatos del archivo GeoJSON. Extrae este valor y asígnalo a la variable `title`.

16-7. Automated Title: In this section, we used the generic title *Global Earthquakes*. Instead, you can use the title for the dataset in the metadata part of the GeoJSON file. Pull this value and assign it to the variable `title`.

16-8. Terremotos recientes: Puedes encontrar en Internet archivos de datos que contienen información sobre los terremotos más recientes en periodos de 1 hora, 1 día, 7 días y 30 días. Entra en <https://earthquake.usgs.gov/earthquakes/feed/v1.0/geojson.php> y verás una lista de enlaces a conjuntos de datos para varios periodos de tiempo, centrados en terremotos de diferentes magnitudes. Descarga uno de estos conjuntos de datos y crea una visualización de la actividad sísmica más reciente.

16-8. Recent Earthquakes: You can find online data files containing information about the most recent earthquakes over 1-hour, 1-day, 7-day, and 30-day periods. Go to <https://earthquake.usgs.gov/earthquakes/feed/v1.0/geojson.php> and you'll see a list of links to datasets for various time periods, focusing on earthquakes of different magnitudes. Download one of these datasets and create a visualization of the most recent earthquake activity.

16-9. Incendios en el Mundo: En los recursos para este capítulo, encontrarás un archivo llamado *world_fires_1_day.csv*. Este archivo contiene información sobre los incendios que arden en diferentes lugares del planeta, incluyendo la latitud, la longitud y la luminosidad de cada incendio. Utilizando el trabajo de procesamiento de datos de la primera parte de este capítulo y el trabajo de cartografía de esta sección, haz un mapa que muestre qué partes del mundo están afectadas por incendios.

16-9. World Fires: In the resources for this chapter, you'll find a file called *world_fires_1_day.csv*. This file contains information about fires burning in different locations around the globe, including the latitude, longitude, and brightness of each fire. Using the data-processing work from the first part of this chapter and the mapping work from this section, make a map that shows which parts of the world are affected by fires.

Puedes descargar versiones más recientes de estos datos en <https://earthdata.nasa.gov/earth-observation-data/near-real-time/firms/active-fire-data>. Puedes encontrar enlaces a los datos en formato CSV en la sección *SHP, KML, and TXT Files*.

You can download more recent versions of this data at <https://earthdata.nasa.gov/earth-observation-data/near-real-time/firms/active-fire-data>. You can find links to the data in CSV format in the *SHP, KML, and TXT Files* section.

Resumen (Summary)

En este capítulo has aprendido a trabajar con conjuntos de datos del mundo real. Procesaste archivos CSV y GeoJSON, y extraíste los datos en los que querías centrarte. Utilizando datos meteorológicos históricos, aprendiste más sobre cómo trabajar con Matplotlib, incluyendo cómo utilizar el módulo `datetime` y cómo trazar múltiples series de datos en un gráfico. Trazaste datos geográficos en un mapa del mundo en Plotly, y aprendiste a personalizar el estilo del mapa.

In this chapter, you learned how to work with real-world datasets. You processed CSV and GeoJSON files, and extracted the data you want to focus on. Using historical weather data, you learned more about working with Matplotlib, including how to use the `datetime` module and how to plot multiple data series on one chart. You plotted geographical data on a world map in Plotly, and learned to customize the style of the map.

A medida que adquieras experiencia trabajando con archivos CSV y JSON, podrás procesar casi cualquier dato que quieras analizar. Puedes descargar la mayoría de los conjuntos de datos en línea en uno de estos formatos o en ambos. Trabajando con estos formatos de `CSV` y `JSON`, también podrás aprender a trabajar con otros formatos de datos más fácilmente.

As you gain experience working with CSV and JSON files, you'll be able to process almost any data you want to analyze. You can download most online datasets in either or both of these formats. By working with these `CSV` and `JSON` formats, you'll be able to learn how to work with other data formats more easily as well.

En el próximo capítulo, escribirás programas que recojan automáticamente sus propios datos de fuentes online, y luego crearás visualizaciones de esos datos. Estas son habilidades divertidas si quieres programar como hobby y son habilidades fundamentales si te interesa programar profesionalmente.

In the next chapter, you'll write programs that automatically gather their own data from online sources, and then you'll create visualizations of that data. These are fun skills to have if you want to program as a hobby and are critical skills if you're interested in programming professionally.

Capítulo 17: Trabajar con API (Chapter 17: Working with APIs)

17

Trabajar con API (17 Working with APIs)



DESCRIPCIÓN DE LA
IMAGEN (En español):

La imagen presenta una ilustración en blanco y negro de una serpiente enroscada alrededor de una turbina o hélice de avión. La serpiente tiene una expresión amigable y lleva gafas. El fondo es de color gris claro, y el contorno de la imagen es circular. No hay texto presente para traducir.

En este capítulo, aprenderás a escribir un programa autónomo que genere una visualización basada en los datos que recupere. Tu programa utilizará una *application programming interface* (*API*) para solicitar automáticamente información específica de un sitio web y luego utilizará esa información para generar una visualización. Como los programas escritos así siempre utilizarán datos actuales para generar una visualización, incluso cuando esos datos puedan estar cambiando rápidamente, la visualización siempre estará actualizada.

In this chapter, you'll learn how to write a self-contained program that generates a visualization based on data it retrieves. Your program will use an *application programming interface* (*API*) to automatically request specific information from a website and then use that information to generate a visualization. Because programs written like this will always use current data to generate a visualization, even when that data might be rapidly changing, the visualization will always be up to date.

Utilizar una API (Using an API)

Una API es una parte de un sitio web diseñada para interactuar con programas. Esos programas utilizan URL muy específicas para solicitar determinada información. Este tipo de solicitud se denomina *API call* . Los datos solicitados se devolverán en un formato

fácilmente procesable, como JSON o CSV. La mayoría de las aplicaciones que utilizan fuentes de datos externas, como las que se integran con sitios de redes sociales, se basan en llamadas a la API.

An API is a part of a website designed to interact with programs. Those programs use very specific URLs to request certain information. This kind of request is called an *API call*. The requested data will be returned in an easily processed format, such as JSON or CSV. Most apps that use external data sources, such as apps that integrate with social media sites, rely on API calls.

Git y GitHub (*Git and GitHub*)

Basaremos nuestra visualización en información de GitHub (<https://github.com>), un sitio que permite a los programadores colaborar en proyectos de codificación. Utilizaremos la API de GitHub para solicitar información sobre proyectos de Python en el sitio, y luego generaremos una visualización interactiva de la popularidad relativa de estos proyectos utilizando Plotly.

We'll base our visualization on information from GitHub (<https://github.com>), a site that allows programmers to collaborate on coding projects. We'll use GitHub's API to request information about Python projects on the site, and then generate an interactive visualization of the relative popularity of these projects using Plotly.

GitHub toma su nombre de Git, un sistema distribuido de control de versiones. Git ayuda a las personas a gestionar su trabajo en un proyecto de forma que se evite que los cambios realizados por una persona interfieran con los cambios que están realizando otras personas. Cuando implementas una nueva función en un proyecto, Git rastrea los cambios que haces en cada archivo. Cuando tu nuevo código funciona, *commit* los cambios que has hecho, y Git registra el nuevo estado de tu proyecto. Si cometes un error y quieres revertir tus cambios, puedes volver fácilmente a cualquier estado que funcionara anteriormente. (Para saber más sobre el control de versiones mediante Git, consulta el Apéndice D .) Los proyectos en GitHub se almacenan en *repositories*, que contiene todo lo relacionado con el proyecto: su código, información sobre sus colaboradores, cualquier incidencia o informe de errores, etc.

GitHub takes its name from Git, a distributed version control system. Git helps people manage their work on a project in a way that prevents changes made by one person from interfering with changes other people are making. When you implement a new feature in a

project, Git tracks the changes you make to each file. When your new code works, you *commit* the changes you've made, and Git records the new state of your project. If you make a mistake and want to revert your changes, you can easily return to any previously working state. (To learn more about version control using Git, see Appendix D .) Projects on GitHub are stored in *repositories* , which contain everything associated with the project: its code, information on its collaborators, any issues or bug reports, and so on.

Cuando a los usuarios de GitHub les gusta un proyecto, pueden ponerle una "estrella" para mostrar su apoyo y hacer un seguimiento de los proyectos que podrían querer utilizar. En este capítulo, escribiremos un programa para descargar automáticamente información sobre los proyectos Python con más estrellas en GitHub, y luego crearemos una visualización informativa de estos proyectos.

When users on GitHub like a project, they can “star” it to show their support and keep track of projects they might want to use. In this chapter, we’ll write a program to automatically download information about the most-starred Python projects on GitHub, and then we’ll create an informative visualization of these projects.

Solicitar datos mediante una llamada a la API (*Requesting Data Using an API Call*)

La API de GitHub te permite solicitar una amplia gama de información a través de llamadas a la API. Para ver cómo es una llamada a la API, introduce lo siguiente en la barra de direcciones de tu navegador y pulsa INTRO:

GitHub’s API lets you request a wide range of information through API calls. To see what an API call looks like, enter the following into your browser’s address bar and press ENTER:

`https://api.github.com/search/repositories?q=language:python+sort:stars`

Esta llamada devuelve el número de proyectos Python alojados actualmente en GitHub, así como información sobre los repositorios Python más populares. Examinemos la llamada. La primera parte, `https://api.github.com/` , dirige la petición a la parte de GitHub que responde a las llamadas a la API. La siguiente parte, `search/repositories` , indica a la API que realice una búsqueda en todos los repositorios de GitHub.

This call returns the number of Python projects currently hosted on GitHub, as well as information about the most popular Python repositories. Let’s examine the call. The first

part, `https://api.github.com/` , directs the request to the part of GitHub that responds to API calls. The next part, `search/repositories` , tells the API to conduct a search through all the repositories on GitHub.

El signo de interrogación después de `repositories` indica que vamos a pasar un argumento. El `q` significa *query* , y el signo igual (`=`) nos permite empezar a especificar una consulta (`q=`). Al utilizar `language:python` , indicamos que sólo queremos información sobre repositorios que tengan Python como lenguaje principal. La parte final, `+sort:stars` , ordena los proyectos por el número de estrellas que han recibido.

The question mark after `repositories` signals that we're about to pass an argument. The `q` stands for *query* , and the equal sign (`=`) lets us begin specifying a query (`q=`). By using `language:python` , we indicate that we want information only on repositories that have Python as the primary language. The final part, `+sort:stars` , sorts the projects by the number of stars they've been given.

El siguiente fragmento muestra las primeras líneas de la respuesta:

The following snippet shows the first few lines of the response:

```
{
❶ "total_count": 8961993,
❷ "incomplete_results": true,
❸ "items": [
  {
    "id": 54346799,
    "node_id": "MDEwOlJlcG9zaXRvcnk1NDM0Njc5OQ==",
    "name": "public-apis",
    "full_name": "public-apis/public-apis",
    --snip--
```

Puedes ver en la respuesta que esta URL no está pensada principalmente para ser introducida por humanos, ya que está en un formato pensado para ser procesado por un programa. GitHub encontró algo menos de nueve millones de proyectos Python en el momento de escribir este artículo ❶ . El valor de `"incomplete_results"` es `true` , lo que nos indica que GitHub no procesó completamente la consulta ❷ . GitHub limita el tiempo de ejecución de cada consulta, para que la API responda a todos los usuarios. En este caso

encontró algunos de los repositorios de Python más populares, pero no tuvo tiempo de encontrarlos todos; lo arreglaremos en un momento. Los "items" devueltos se muestran en la lista siguiente, que contiene detalles sobre los proyectos Python más populares en GitHub ❸ .

You can see from the response that this URL is not primarily intended to be entered by humans, because it's in a format that's meant to be processed by a program. GitHub found just under nine million Python projects as of this writing ❶ . The value for "incomplete_results" is true , which tells us that GitHub didn't fully process the query ❷ . GitHub limits how long each query can run, in order to keep the API responsive for all users. In this case it found some of the most popular Python repositories, but it didn't have time to find all of them; we'll fix that in a moment. The "items" returned are displayed in the list that follows, which contains details about the most popular Python projects on GitHub ❸ .

Instalar solicitudes (*Installing Requests*)

El paquete *Requests* permite a un programa Python solicitar fácilmente información a un sitio web y examinar la respuesta. Utiliza pip para instalar Solicitudes:

The *Requests* package allows a Python program to easily request information from a website and examine the response. Use pip to install Requests:

```
$ python -m pip install --user requests
```

Si utilizas un comando distinto de `python` para ejecutar programas o iniciar una sesión de terminal, como `python3` , tu comando tendrá el siguiente aspecto:

If you use a command other than `python` to run programs or start a terminal session, such as `python3` , your command will look like this:

```
$ python3 -m pip install --user requests
```

Procesar una respuesta de la API (*Processing an API Response*)

Ahora escribiremos un programa para realizar automáticamente una llamada a la API y procesar los resultados:

Now we'll write a program to automatically issue an API call and process the results:

python_repos.py

python_repos.py

```
import requests

# Make an API call and check the response.
❶ url = "https://api.github.com/search/repositories"
url += "?q=language:python+sort:stars+stars:>10000"

❷ headers = {"Accept": "application/vnd.github.v3+json"}
❸ r = requests.get(url, headers=headers)
❹ print(f'Status code: {r.status_code}')

# Convert the response object to a dictionary.
❺ response_dict = r.json()

# Process results.
print(response_dict.keys())
```

Primero importamos el módulo `requests` . Luego asignamos la URL de la llamada a la API a la variable `url` ❶ . Se trata de una URL larga, así que la dividimos en dos líneas. La primera línea es la parte principal de la URL, y la segunda es la cadena de consulta. Hemos incluido una condición más a la cadena de consulta original: `stars:>10000` , que indica a GitHub que sólo busque repositorios de Python que tengan más de 10.000 estrellas. Esto debería permitir a GitHub devolver un conjunto de resultados completo y coherente.

We first import the `requests` module. Then we assign the URL of the API call to the `url` variable ❶ . This is a long URL, so we break it into two lines. The first line is the main part of the URL, and the second line is the query string. We've included one more condition to the original query string: `stars:>10000` , which tells GitHub to only look for Python repositories that have more than 10,000 stars. This should allow GitHub to return a complete, consistent set of results.

GitHub está actualmente en la tercera versión de su API, así que definimos cabeceras para la llamada a la API que piden explícitamente utilizar esta versión de la API, y devuelven los resultados en el formato JSON ❷ . A continuación, utilizamos `requests` para realizar la llamada a la API ❸ . Llamamos a `get()` y le pasamos la URL y la cabecera que hemos definido, y asignamos el objeto respuesta a la variable `r` .

GitHub is currently on the third version of its API, so we define headers for the API call that ask explicitly to use this version of the API, and return the results in the JSON format ❷ . Then we use `requests` to make the call to the API ❸ . We call `get()` and pass it the URL and the header that we defined, and we assign the response object to the variable `r` .

El objeto respuesta tiene un atributo llamado `status_code` , que nos indica si la solicitud se ha realizado correctamente. (Un código de estado 200 indica una respuesta correcta). Imprimimos el valor de `status_code` para asegurarnos de que la llamada se ha realizado correctamente ❹ . Pedimos a la API que devolviera la información en formato JSON, así que utilizamos el método `json()` para convertir la información en un diccionario Python ❺ . Asignamos el diccionario resultante a `response_dict` .

The response object has an attribute called `status_code` , which tells us whether the request was successful. (A status code of 200 indicates a successful response.) We print the value of `status_code` so we can make sure the call went through successfully ❹ . We asked the API to return the information in JSON format, so we use the `json()` method to convert the information to a Python dictionary ❺ . We assign the resulting dictionary to `response_dict` .

Por último, imprimimos las claves de `response_dict` y vemos la siguiente salida:

Finally, we print the keys from `response_dict` and see the following output:

```
Status code: 200
dict_keys(['total_count', 'incomplete_results', 'items'])
```

Como el código de estado es 200 , sabemos que la solicitud se ha realizado correctamente. El diccionario de respuesta sólo contiene tres claves: `'total_count'` , `'incomplete_results'` , y `'items'` . Echemos un vistazo al diccionario de respuesta.

Because the status code is 200 , we know that the request was successful. The response dictionary contains only three keys: 'total_count' , 'incomplete_results' , and 'items' . Let's take a look inside the response dictionary.

Trabajar con el diccionario de respuesta *(Working with the Response Dictionary)*

Con la información de la llamada a la API representada como un diccionario, podemos trabajar con los datos allí almacenados. Vamos a generar alguna salida que resuma la información. Es una buena forma de asegurarnos de que hemos recibido la información que esperábamos y de empezar a examinar la información que nos interesa:

With the information from the API call represented as a dictionary, we can work with the data stored there. Let's generate some output that summarizes the information. This is a good way to make sure we received the information we expected, and to start examining the information we're interested in:

python_repos.py

python_repos.py

```
import requests
```

```
# Make an API call and store the response.
```

```
--snip--
```

```
# Convert the response object to a dictionary.
```

```
response_dict = r.json()
```

```
❶ print(f"Total repositories: {response_dict['total_count']}")
```

```
print(f"Complete results: {not response_dict['incomplete_results']}")
```

```
# Explore information about the repositories.
```

```
❷ repo_dicts = response_dict['items']
```

```
print(f"Repositories returned: {len(repo_dicts)}")
```

```
# Examine the first repository.
```

```
③ repo_dict = repo_dicts[0]
④ print(f"\nKeys: {len(repo_dict)}")
⑤ for key in sorted(repo_dict.keys()):
    print(key)
```

Empezamos a explorar el diccionario de respuestas imprimiendo el valor asociado a 'total_count' , que representa el número total de repositorios de Python devueltos por esta llamada a la API ❶ . También utilizamos el valor asociado a 'incomplete_results' , para saber si GitHub pudo procesar completamente la consulta. En lugar de imprimir este valor directamente, imprimimos su opuesto: un valor de True indicará que hemos recibido un conjunto completo de resultados.

We start exploring the response dictionary by printing the value associated with 'total_count' , which represents the total number of Python repositories returned by this API call ❶ . We also use the value associated with 'incomplete_results' , so we'll know if GitHub was able to fully process the query. Rather than printing this value directly, we print its opposite: a value of True will indicate that we received a complete set of results.

El valor asociado a 'items' es una lista que contiene una serie de diccionarios, cada uno de los cuales contiene datos sobre un repositorio Python individual. Asignamos esta lista de diccionarios a repo_dicts ❷ . A continuación, imprimimos la longitud de repo_dicts para ver de cuántos repositorios tenemos información.

The value associated with 'items' is a list containing a number of dictionaries, each of which contains data about an individual Python repository. We assign this list of dictionaries to repo_dicts ❷ . We then print the length of repo_dicts to see how many repositories we have information for.

Para ver más de cerca la información devuelta sobre cada repositorio, sacamos el primer elemento de repo_dicts y lo asignamos a repo_dict ❸ . A continuación, imprimimos el número de claves del diccionario para ver cuánta información tenemos ❹ . Por último, imprimimos todas las claves del diccionario para ver qué tipo de información incluye ❺ .

To look closer at the information returned about each repository, we pull out the first item from repo_dicts and assign it to repo_dict ❸ . We then print the number of keys in the dictionary to see how much information we have ❹ . Finally, we print all the dictionary's keys to see what kind of information is included ❺ .

Los resultados nos dan una idea más clara de los datos reales:

The results give us a clearer picture of the actual data:

Status code: 200

❶ Total repositories: 248

❷ Complete results: True

Repositories returned: 30

❸ Keys: 78

allow_forking

archive_url

archived

--snip--

url

visiblity

watchers

watchers_count

En el momento de escribir esto, sólo hay 248 repositorios de Python con más de 10.000 estrellas ❶ . Podemos ver que GitHub fue capaz de procesar completamente la llamada a la API ❷ . En esta respuesta, GitHub devolvió información sobre los primeros 30 repositorios que coinciden con las condiciones de nuestra consulta. Si queremos más repositorios, podemos solicitar páginas adicionales de datos.

At the time of this writing, there are only 248 Python repositories with over 10,000 stars ❶ . We can see that GitHub was able to fully process the API call ❷ . In this response, GitHub returned information about the first 30 repositories that match the conditions of our query. If we want more repositories, we can request additional pages of data.

La API de GitHub devuelve mucha información sobre cada repositorio: hay 78 claves en `repo_dict` ❸ . Cuando eches un vistazo a estas claves, te harás una idea del tipo de información que puedes extraer sobre un proyecto. (La única forma de saber qué información está disponible a través de una API es leer la documentación o examinar la información a través del código, como estamos haciendo aquí).

GitHub's API returns a lot of information about each repository: there are 78 keys in `repo_dict` ❸ . When you look through these keys, you'll get a sense of the kind of information you can extract about a project. (The only way to know what information is available through an API is to read the documentation or to examine the information through code, as we're doing here.)

Vamos a extraer los valores de algunas de las claves de `repo_dict` :

Let's pull out the values for some of the keys in `repo_dict` :

python_repos.py

python_repos.py

```
--snip--
# Examine the first repository.
repo_dict = repo_dicts[0]

print("\nSelected information about first repository:")
❶ print(f'Name: {repo_dict['name']}')
❷ print(f'Owner: {repo_dict['owner']['login']}')
❸ print(f'Stars: {repo_dict['stargazers_count']}')
print(f'Repository: {repo_dict['html_url']}')
❹ print(f'Created: {repo_dict['created_at']}')
❺ print(f'Updated: {repo_dict['updated_at']}')
print(f'Description: {repo_dict['description']}')
```

Aquí imprimimos los valores de varias claves del diccionario del primer repositorio. Empezamos con el nombre del proyecto ❶ . Un diccionario entero representa al propietario del proyecto, así que utilizamos la clave `owner` para acceder al diccionario que representa al propietario, y luego utilizamos la clave `login` para obtener el nombre de usuario del propietario ❷ . A continuación, imprimimos cuántas estrellas ha obtenido el proyecto ❸ y la URL del repositorio GitHub del proyecto. A continuación, mostramos cuándo se creó ❹ y cuándo se actualizó por última vez ❺ . Por último, mostramos la descripción del repositorio.

Here, we print the values for a number of keys from the first repository's dictionary. We start with the name of the project ❶ . An entire dictionary represents the project's owner, so we use the key `owner` to access the dictionary representing the owner, and then use the key `login` to get the owner's login name ❷ . Next, we print how many stars the project has earned ❸ and the URL for the project's GitHub repository. We then show when it was created ❹ and when it was last updated ❺ . Finally, we print the repository's description.

El resultado debería ser algo parecido a esto:

The output should look something like this:

```
Status code: 200
Total repositories: 248
Complete results: True
Repositories returned: 30

Selected information about first repository:
Name: public-apis
Owner: public-apis
Stars: 191493
Repository: https://github.com/public-apis/public-apis
Created: 2016-03-20T23:49:42Z
Updated: 2022-05-12T06:37:11Z
Description: A collective list of free APIs
```

Podemos ver que el proyecto Python con más estrellas en GitHub en el momento de escribir esto es *public-apis* . Su propietario es una organización con el mismo nombre, y ha recibido estrellas de casi 200.000 usuarios de GitHub. Podemos ver la URL del repositorio del proyecto, su fecha de creación en marzo de 2016, y que se ha actualizado recientemente. Además, la descripción nos dice que *public-apis* contiene una lista de API gratuitas que podrían interesar a los programadores.

We can see that the most-starred Python project on GitHub as of this writing is *public-apis* . Its owner is an organization with the same name, and it has been starred by almost 200,000 GitHub users. We can see the URL for the project's repository, its creation date of March 2016, and that it was updated recently. Additionally, the description tells us that *public-apis* contains a list of free APIs that programmers might be interested in.

Resumen de los principales repositorios *(Summarizing the Top Repositories)*

Cuando hagamos una visualización de estos datos, queremos incluir más de un repositorio. Escribamos un bucle para imprimir la información seleccionada sobre cada repositorio que devuelva la llamada a la API, de forma que podamos incluirlos todos en la visualización:

When we make a visualization for this data, we'll want to include more than one repository. Let's write a loop to print selected information about each repository the API call returns so we can include them all in the visualization:

python_repos.py

python_repos.py

```
--snip--
# Explore information about the repositories.
repo_dicts = response_dict['items']
print(f"Repositories returned: {len(repo_dicts)}")

❶ print("\nSelected information about each repository:")
❷ for repo_dict in repo_dicts:
    print(f"\nName: {repo_dict['name']}")
    print(f"Owner: {repo_dict['owner']['login']}")
    print(f"Stars: {repo_dict['stargazers_count']}")
    print(f"Repository: {repo_dict['html_url']}")
    print(f>Description: {repo_dict['description']}")
```

Primero imprimimos un mensaje introductorio ❶ . A continuación, recorremos en bucle todos los diccionarios de `repo_dicts` ❷ . Dentro del bucle, imprimimos el nombre de cada proyecto, su propietario, cuántas estrellas tiene, su URL en GitHub y la descripción del proyecto:

We first print an introductory message ❶ . Then we loop through all the dictionaries in `repo_dicts` ❷ . Inside the loop, we print the name of each project, its owner, how many stars it has, its URL on GitHub, and the project's description:

Status code: 200
Total repositories: 248
Complete results: True
Repositories returned: 30

Selected information about each repository:

Name: public-apis
Owner: public-apis
Stars: 191494
Repository: <https://github.com/public-apis/public-apis>
Description: A collective list of free APIs

Name: system-design-primer
Owner: donnemartin
Stars: 179952
Repository: <https://github.com/donnemartin/system-design-primer>
Description: Learn how to design large-scale systems. Prep for the system design interview. Includes Anki flashcards.
--snip--

Name: PayloadsAllTheThings
Owner: swisskyrepo
Stars: 37227
Repository: <https://github.com/swisskyrepo/PayloadsAllTheThings>
Description: A list of useful payloads and bypass for Web Application Security and Pentest/CTF

En estos resultados aparecen algunos proyectos interesantes, y puede que merezca la pena echar un vistazo a unos cuantos. Pero no pases demasiado tiempo aquí, porque estamos a punto de crear una visualización que hará que los resultados sean mucho más fáciles de leer.

Some interesting projects appear in these results, and it might be worth looking at a few. But don't spend too much time here, because we're about to create a visualization that will make the results much easier to read.

Supervisar los límites de velocidad de la API (Monitoring API Rate Limits)

La mayoría de las API tienen *rate limits*, lo que significa que hay un límite en el número de peticiones que puedes hacer en un determinado periodo de tiempo. Para ver si te estás acercando a los límites de GitHub, introduce https://api.github.com/rate_limit en un navegador web. Deberías ver una respuesta que empiece así:

Most APIs have *rate limits*, which means there's a limit to how many requests you can make in a certain amount of time. To see if you're approaching GitHub's limits, enter https://api.github.com/rate_limit into a web browser. You should see a response that begins like this:

```
{
  "resources": {
    --snip--
    ❶ "search": {
    ❷   "limit": 10,
    ❸   "remaining": 9,
    ❹   "reset": 1652338832,
      "used": 1,
      "resource": "search"
    },
    --snip--
```

La información que nos interesa es el límite de solicitudes para la API de búsqueda ❶. Vemos que el límite es de 10 peticiones por minuto ❷ y que nos quedan 9 peticiones para el minuto actual ❸. El valor asociado a la clave "reset" representa el tiempo en *Unix* o *epoch time* (el número de segundos desde la medianoche del 1 de enero de 1970) en que se restablecerá nuestra cuota ❹. Si alcanzas tu cuota, recibirás una breve respuesta que te informará de que has llegado al límite de la API. Si alcanzas el límite, espera a que se restablezca tu cuota.

The information we're interested in is the rate limit for the search API ❶. We see that the limit is 10 requests per minute ❷ and that we have 9 requests remaining for the current minute ❸. The value associated with the key "reset" represents the time in *Unix* or *epoch time* (the number of seconds since midnight on January 1, 1970) when our quota will reset ❹. If you reach your quota, you'll get a short response that lets you know you've reached the API limit. If you reach the limit, just wait until your quota resets.

Nota (Note)

Muchas API requieren que te registres y obtengas una clave API o un token de acceso para realizar llamadas a la API. En el momento de escribir esto, GitHub no tiene ese requisito, pero si obtienes un token de acceso, tus límites serán mucho mayores.

Many APIs require you to register and obtain an API key or access token to make API calls. As of this writing, GitHub has no such requirement, but if you obtain an access token, your limits will be much higher.

Visualizar repositorios con Plotly (*Visualizing Repositories Using Plotly*)

Vamos a hacer una visualización utilizando los datos que hemos recopilado para mostrar la popularidad relativa de los proyectos Python en GitHub. Haremos un gráfico de barras interactivo: la altura de cada barra representará el número de estrellas que ha adquirido el proyecto, y podrás hacer clic en la etiqueta de la barra para ir a la página principal de ese proyecto en GitHub.

Let's make a visualization using the data we've gathered to show the relative popularity of Python projects on GitHub. We'll make an interactive bar chart: the height of each bar will represent the number of stars the project has acquired, and you'll be able to click the bar's label to go to that project's home on GitHub.

Guarda una copia del programa en el que hemos estado trabajando como `python_repos_visual.py`, y luego modifícalo para que quede como sigue:

Save a copy of the program we've been working on as `python_repos_visual.py`, then modify it so it reads as follows:

`python_repos_visual.py`

`python_repos_visual.py`

```
import requests
import plotly.express as px

# Make an API call and check the response.
url = "https://api.github.com/search/repositories"
url += "?q=language:python+sort:stars+stars:>10000"

headers = {"Accept": "application/vnd.github.v3+json"}
r = requests.get(url, headers=headers)
❶ print(f'Status code: {r.status_code}')

# Process overall results.
response_dict = r.json()
❷ print(f'Complete results: {not response_dict['incomplete_results']}')

# Process repository information.
repo_dicts = response_dict['items']
❸ repo_names, stars = [], []
for repo_dict in repo_dicts:
    repo_names.append(repo_dict['name'])
    stars.append(repo_dict['stargazers_count'])

# Make visualization.
❹ fig = px.bar(x=repo_names, y=stars)
fig.show()
```

Importamos Plotly Express y realizamos la llamada a la API como venimos haciendo. Seguimos imprimiendo el estado de la respuesta de la llamada a la API para saber si hay algún problema ❶. Cuando procesemos los resultados globales, seguimos imprimiendo el mensaje que confirma que hemos obtenido un conjunto completo de resultados ❷. Eliminamos el resto de las llamadas a `print()` porque ya no estamos en la fase exploratoria; sabemos que tenemos los datos que queremos.

We import Plotly Express and then make the API call as we have been doing. We continue to print the status of the API call response so we'll know if there is a problem ❶. When we process the overall results, we continue to print the message confirming that we got a complete set of results ❷. We remove the rest of the `print()` calls because we're no longer in the exploratory phase; we know we have the data we want.

A continuación, creamos dos listas vacías ❸ para almacenar los datos que incluiremos en el gráfico inicial. Necesitaremos el nombre de cada proyecto para etiquetar las barras (`repo_names`) y el número de estrellas para determinar la altura de las barras (`stars`). En el bucle, añadimos a estas listas el nombre de cada proyecto y el número de estrellas que tiene.

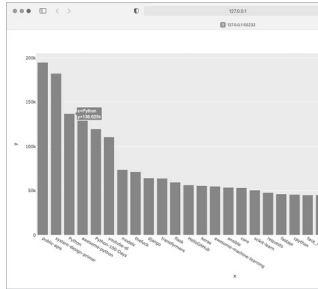
We then create two empty lists ❸ to store the data we'll include in the initial chart. We'll need the name of each project to label the bars (`repo_names`) and the number of stars to determine the height of the bars (`stars`). In the loop, we append the name of each project and the number of stars it has to these lists.

Hacemos la visualización inicial con sólo dos líneas de código ❹ . Esto es coherente con la filosofía de Plotly Express de que deberías poder ver tu visualización lo más rápidamente posible antes de refinar su aspecto. Aquí utilizamos la función `px.bar()` para crear un gráfico de barras. Pasamos la lista `repo_names` como argumento `x` y `stars` como argumento `y` .

We make the initial visualization with just two lines of code ❹ . This is consistent with Plotly Express's philosophy that you should be able to see your visualization as quickly as possible before refining its appearance. Here we use the `px.bar()` function to create a bar chart. We pass the list `repo_names` as the `x` argument and `stars` as the `y` argument.

La [Figura 17-1](#) muestra el gráfico resultante. Podemos ver que los primeros proyectos son significativamente más populares que el resto, pero todos ellos son proyectos importantes en el ecosistema Python.

[Figure 17-1](#) shows the resulting chart. We can see that the first few projects are significantly more popular than the rest, but all of them are important projects in the Python ecosystem.



DESCRIPCIÓN DE LA IMAGEN (En español):

En la imagen se muestra un gráfico de barras con datos referentes a diferentes lenguajes de programación. En el eje horizontal, etiquetado como "x", se enumeran varios lenguajes:

JavaScript, Python, Java, TypeScript, C#, C++, PHP, C, Shell, Ruby, Swift, Go, Kotlin, Rust, Objective-C, Dart, Groovy, SQL y Bash/shell. En el eje vertical, etiquetado como "y", se representan valores numéricos que van incrementándose de 50k en 50k, alcanzando hasta 200k. Las barras indican la popularidad o la medida relacionada de cada lenguaje, siendo JavaScript el que tiene la barra más alta, seguido por Python y otros lenguajes en orden decreciente. Adicionalmente, se observa que al pasar el cursor sobre la barra de Python, se muestra un texto emergente que dice: "Python 167.526".

Figura 17-1 : Los proyectos Python con más estrellas en GitHub

Figure 17-1 : The most-starred Python projects on GitHub

Estilizar el gráfico (Styling the Chart)

Plotly admite varias formas de dar estilo y personalizar los gráficos, una vez que sabes que la información del gráfico es correcta. Haremos algunos cambios en la llamada inicial a `px.bar()` y luego realizaremos otros ajustes en el objeto `fig` una vez creado.

Plotly supports a number of ways to style and customize the plots, once you know the information in the plot is correct. We'll make some changes in the initial `px.bar()` call and then make some further adjustments to the `fig` object after it's been created.

Empezaremos a dar estilo al gráfico añadiendo un título y etiquetas para cada eje:

We'll start styling the chart by adding a title and labels for each axis:

python_repos_visual.py

python_repos_visual.py

```
--snip--
# Make visualization.
title = "Most-Starred Python Projects on GitHub"
labels = {'x': 'Repository', 'y': 'Stars'}
fig = px.bar(x=repo_names, y=stars, title=title, labels=labels)

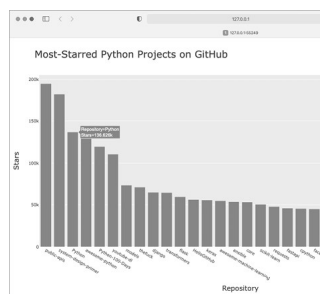
❶ fig.update_layout(title_font_size=28, xaxis_title_font_size=20,
                    yaxis_title_font_size=20)

fig.show()
```

Primero añadimos un título y etiquetas para cada eje, como hicimos en los Capítulos 15 y 16 . A continuación, utilizamos el método `fig.update_layout()` para modificar elementos específicos del gráfico ❶ . Plotly utiliza una convención en la que los aspectos de un elemento del gráfico se conectan mediante guiones bajos. A medida que te familiarices con la documentación de Plotly, empezarás a ver patrones consistentes en cómo se nombran y modifican los distintos elementos de un gráfico. Aquí fijamos el tamaño de la fuente del título en 28 y el tamaño de la fuente del título de cada eje en 20 . El resultado se muestra

en la [Figura 17-2](#) .

We first add a title and labels for each axis, as we did in Chapters 15 and 16 . We then use the `fig.update_layout()` method to modify specific elements of the chart ❶ . Plotly uses a convention where aspects of a chart element are connected by underscores. As you become familiar with Plotly's documentation, you'll start to see consistent patterns in how different elements of a chart are named and modified. Here we set the title font size to 28 and the font size for each axis title to 20 . The result is shown in [Figure 17-2](#) .



DESCRIPCIÓN DE LA
IMAGEN (En español):

La imagen contiene un gráfico de barras titulado "Proyectos de Python con Más Estrellas en GitHub". En el eje horizontal, titulado "Repositorio", se enumeran varios proyectos de Python. En el eje vertical, titulado "Estrellas", se muestra la cantidad de estrellas que cada proyecto ha recibido. Cada barra representa un proyecto de Python en GitHub, y su altura refleja el número de estrellas acumuladas, siendo el repositorio "Python" el que tiene más de 150k estrellas.

Figura 17-2 : Se ha añadido un título al gráfico principal, y también a cada eje.

Figure 17-2 : A title has been added to the main chart, and to each axis as well.

Añadir información sobre herramientas personalizada (Adding Custom Tooltips)

En Plotly, puedes situar el cursor sobre una barra individual para mostrar la información que representa. Esto se denomina comúnmente *tooltip* , y en este caso, actualmente muestra el número de estrellas que tiene un proyecto. Vamos a crear una información sobre herramientas personalizada para mostrar la descripción de cada proyecto, así como

su propietario.

In Plotly, you can hover the cursor over an individual bar to show the information the bar represents. This is commonly called a *tooltip*, and in this case, it currently shows the number of stars a project has. Let's create a custom tooltip to show each project's description as well as the project's owner.

Necesitamos obtener algunos datos adicionales para generar la información sobre herramientas:

We need to pull some additional data to generate the tooltips:

python_repos_visual.py

python_repos_visual.py

```
--snip--
# Process repository information.
repo_dicts = response_dict['items']
❶ repo_names, stars, hover_texts = [], [], []
for repo_dict in repo_dicts:
    repo_names.append(repo_dict['name'])
    stars.append(repo_dict['stargazers_count'])

    # Build hover texts.
    ❷ owner = repo_dict['owner']['login']
    description = repo_dict['description']
    ❸ hover_text = f"{owner}<br />{description}"
    hover_texts.append(hover_text)

# Make visualization.
title = "Most-Starred Python Projects on GitHub"
labels = {'x': 'Repository', 'y': 'Stars'}
❹ fig = px.bar(x=repo_names, y=stars, title=title, labels=labels,
               hover_name=hover_texts)
```



```
fig.update_layout(title_font_size=28, xaxis_title_font_size=20,  
                  yaxis_title_font_size=20)
```

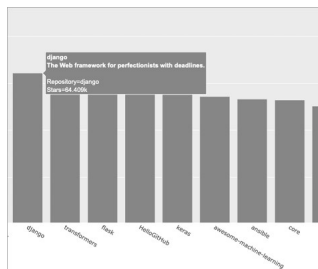
```
fig.show()
```

Primero definimos una nueva lista vacía, `hover_texts`, para guardar el texto que queremos mostrar de cada proyecto ❶. En el bucle donde procesamos los datos, extraemos el propietario y la descripción de cada proyecto ❷. Plotly te permite utilizar código HTML dentro de los elementos de texto, así que generamos una cadena para la etiqueta con un salto de línea (`<br />`) entre el nombre de usuario del propietario del proyecto y la descripción ❸. A continuación, añadimos esta etiqueta a la lista `hover_texts`.

We first define a new empty list, `hover_texts`, to hold the text we want to display for each project ❶. In the loop where we process the data, we pull the owner and the description for each project ❷. Plotly allows you to use HTML code within text elements, so we generate a string for the label with a line break (`<br />`) between the project owner's username and the description ❸. We then append this label to the list `hover_texts`.

En la llamada a `px.bar()`, añadimos el argumento `hover_name` y le pasamos `hover_texts` ❹. Este es el mismo enfoque que utilizamos para personalizar la etiqueta de cada punto en el mapa de la actividad sísmica mundial. A medida que Plotly crea cada barra, extraerá etiquetas de esta lista y sólo las mostrará cuando el visor pase el ratón por encima de una barra. [La Figura 17-3](#) muestra uno de estos tooltips personalizados.

In the `px.bar()` call, we add the `hover_name` argument and pass it `hover_texts` ❹. This is the same approach we used to customize the label for each dot in the map of global earthquake activity. As Plotly creates each bar, it will pull labels from this list and only display them when the viewer hovers over a bar. [Figure 17-3](#) shows one of these custom tooltips.



DESCRIPCIÓN DE LA
IMAGEN (En español):

En la imagen se muestra un gráfico de barras con varios marcos de software y proyectos, cada uno acompañado de sus estrellas en el repositorio. Una barra

destacada corresponde a "django" con el lema "El marco web para perfeccionistas con plazos". El repositorio se indica como "django" y tiene 64,409 estrellas. Las otras barras, de izquierda a derecha, corresponden a: "flask", "transformers", "HeliogihuB", "keras", "awesome-machine learning", "ansible", "core", "scikit-learn" y "requests". Las barras tienen alturas similares, excepto la primera que está resaltada.

Figura 17-3 : Al pasar el ratón por encima de una barra se muestra el propietario y la descripción del proyecto.

Figure 17-3 : Hovering over a bar shows the project's owner and description.

Añadir enlaces clicables (*Adding Clickable Links*)

Como Plotly te permite utilizar HTML en elementos de texto, podemos añadir fácilmente enlaces a un gráfico. Utilicemos las etiquetas del eje x para que el espectador pueda visitar la página de inicio de cualquier proyecto en GitHub. Tenemos que extraer las URL de los datos y utilizarlas al generar las etiquetas del eje x :

Because Plotly allows you to use HTML on text elements, we can easily add links to a chart. Let's use the x -axis labels as a way to let the viewer visit any project's home page on GitHub. We need to pull the URLs from the data and use them when generating the x -axis labels:

python_repos_visual.py

python_repos_visual.py

```
--snip--
# Process repository information.
repo_dicts = response_dict['items']
❶ repo_links, stars, hover_texts = [], [], []
for repo_dict in repo_dicts:
    # Turn repo names into active links.
    repo_name = repo_dict['name']
    ❷ repo_url = repo_dict['html_url']
    ❸ repo_link = f"<a href='{repo_url}'>{repo_name}</a>"
    repo_links.append(repo_link)

    stars.append(repo_dict['stargazers_count'])
--snip--

# Make visualization.
title = "Most-Starred Python Projects on GitHub"
labels = {'x': 'Repository', 'y': 'Stars'}
fig = px.bar(x=repo_links, y=stars, title=title, labels=labels,
             hover_name=hover_texts)

fig.update_layout(title_font_size=28, xaxis_title_font_size=20,
                  yaxis_title_font_size=20)

fig.show()
```

Actualizamos el nombre de la lista que estamos creando de `repo_names` a `repo_links` para comunicar con mayor precisión el tipo de información que estamos reuniendo para el gráfico ❶ . A continuación, extraemos la URL del proyecto de `repo_dict` y la asignamos a la variable temporal `repo_url` ❷ . A continuación, generamos un enlace al proyecto ❸ . Utilizamos la etiqueta de anclaje HTML, que tiene la forma `<a href='URL'>link text</a>` , para generar el enlace. A continuación, añadimos este enlace a `repo_links` .

We update the name of the list we're creating from `repo_names` to `repo_links` to more accurately communicate the kind of information we're putting together for the chart ❶ . We then pull the URL for the project from `repo_dict` and assign it to the temporary variable `repo_url` ❷ . Next, we generate a link to the project ❸ . We use the HTML anchor tag, which has the form `<a href='URL'>link text</a>` , to generate the link. We then append this link to `repo_links` .

Cuando llamamos a `px.bar()` , utilizamos `repo_links` para los valores de `x` en el gráfico. El resultado es el mismo que antes, pero ahora el espectador puede hacer clic en cualquiera de los nombres de proyecto de la parte inferior del gráfico para visitar la página principal de ese proyecto en GitHub. ¡Ahora tenemos una visualización interactiva e informativa de datos obtenidos a través de una API!

When we call `px.bar()` , we use `repo_links` for the `x` -values in the chart. The result looks the same as before, but now the viewer can click any of the project names at the bottom of the chart to visit that project's home page on GitHub. Now we have an interactive, informative visualization of data retrieved through an API!

Personalizar los colores de los marcadores *(Customizing Marker Colors)*

Una vez creado un gráfico, se puede personalizar casi cualquier aspecto del gráfico mediante un método de actualización. Ya hemos utilizado anteriormente el método `update_layout()` . Otro método, `update_traces()` , puede utilizarse para personalizar los datos que se representan en un gráfico.

Once a chart has been created, almost any aspect of the chart can be customized through an update method. We've used the `update_layout()` method previously. Another method, `update_traces()` , can be used to customize the data that's represented on a chart.

Cambiamos las barras a un azul más oscuro, con cierta transparencia:

Let's change the bars to a darker blue, with some transparency:

```
--snip--
fig.update_layout(title_font_size=28, xaxis_title_font_size=20,
                  yaxis_title_font_size=20)

fig.update_traces(marker_color='SteelBlue', marker_opacity=0.6)

fig.show()
```

En Plotly, un *trace* se refiere a una colección de datos en un gráfico. El método `update_traces()` puede recibir varios argumentos diferentes; cualquier argumento que

empiece por `marker_` afecta a los marcadores del gráfico. Aquí establecemos el color de cada marcador en 'SteelBlue' ; cualquier color CSS con nombre funcionará aquí. También establecemos la opacidad de cada marcador en 0.6 . Una opacidad de 1,0 será totalmente opaca, y una opacidad de 0 será totalmente invisible.

In Plotly, a *trace* refers to a collection of data on a chart. The `update_traces()` method can take a number of different arguments; any argument that starts with `marker_` affects the markers on the chart. Here we set each marker's color to 'SteelBlue' ; any named CSS color will work here. We also set the opacity of each marker to 0.6 . An opacity of 1.0 will be entirely opaque, and an opacity of 0 will be entirely invisible.

Más información sobre Plotly y la API de GitHub *(More About Plotly and the GitHub API)*

La documentación de Plotly es extensa y está bien organizada; sin embargo, puede ser difícil saber por dónde empezar a leer. Un buen lugar para empezar es el artículo "Plotly Express en Python", en <https://plotly.com/python/plotly-express> . Es una visión general de todos los gráficos que puedes hacer con Plotly Express, y puedes encontrar enlaces a artículos más largos sobre cada tipo de gráfico individual.

Plotly's documentation is extensive and well organized; however, it can be hard to know where to start reading. A good place to start is with the article "Plotly Express in Python," at <https://plotly.com/python/plotly-express> . This is an overview of all the plots you can make with Plotly Express, and you can find links to longer articles about each individual chart type.

Si quieres entender cómo personalizar mejor los gráficos de Plotly, el artículo "Estilizar gráficos de Plotly Express en Python" ampliará lo que has visto en los capítulos 15- 17 . Puedes encontrar este artículo en <https://plotly.com/python/styling-plotly-express> .

If you want to understand how to customize Plotly charts better, the article "Styling Plotly Express Figures in Python" will expand on what you've seen in Chapters 15 – 17 . You can find this article at <https://plotly.com/python/styling-plotly-express> .

Para más información sobre la API de GitHub, consulta su documentación en <https://docs.github.com/en/rest> . Aquí aprenderás a extraer una amplia variedad de información de GitHub. Para ampliar lo que has visto en este proyecto, busca la sección

Buscar de la referencia en la barra lateral. Si tienes una cuenta de GitHub, puedes trabajar con tus propios datos, así como con los datos disponibles públicamente de los repositorios de otros usuarios.

For more about the GitHub API, refer to its documentation at <https://docs.github.com/en/rest>. Here you'll learn how to pull a wide variety of information from GitHub. To expand on what you saw in this project, look for the Search section of the reference in the sidebar. If you have a GitHub account, you can work with your own data as well as the publicly available data from other users' repositories.

La API de Hacker News (*The Hacker News API*)

Para explorar cómo utilizar las llamadas a la API en otros sitios, echemos un vistazo rápido a Hacker News (<https://news.ycombinator.com>). En Hacker News, la gente comparte artículos sobre programación y tecnología y participa en animados debates sobre esos artículos. La API de Hacker News proporciona acceso a datos sobre todos los envíos y comentarios del sitio, y puedes utilizar la API sin tener que registrarte para obtener una clave.

To explore how to use API calls on other sites, let's take a quick look at Hacker News (<https://news.ycombinator.com>). On Hacker News, people share articles about programming and technology and engage in lively discussions about those articles. The Hacker News API provides access to data about all submissions and comments on the site, and you can use the API without having to register for a key.

La siguiente llamada devuelve información sobre el artículo principal actual en el momento de escribir este artículo:

The following call returns information about the current top article as of this writing:

`https://hacker-news.firebaseio.com/v0/item/31353677.json`

Cuando introduzcas esta URL en un navegador, verás que el texto de la página está encerrado entre llaves, lo que significa que es un diccionario. Pero la respuesta es difícil de examinar sin un formato mejor. Pasemos esta URL por el método `json.dumps()`, como hicimos en el proyecto terremoto del Capítulo 16, para poder explorar el tipo de información que se devuelve sobre un artículo:

When you enter this URL in a browser, you'll see that the text on the page is enclosed by braces, meaning it's a dictionary. But the response is difficult to examine without some better formatting. Let's run this URL through the `json.dumps()` method, like we did in the earthquake project in Chapter 16 , so we can explore the kind of information that's returned about an article:

hn_articulo.py

hn_article.py

```
import requests
import json

# Make an API call, and store the response.
url = "https://hacker-news.firebaseio.com/v0/item/31353677.json"
r = requests.get(url)
print(f"Status code: {r.status_code}")

# Explore the structure of the data.
response_dict = r.json()
response_string = json.dumps(response_dict, indent=4)
❶ print(response_string)
```

Todo en este programa debería resultarte familiar, porque lo hemos utilizado todo en los dos capítulos anteriores. La principal diferencia aquí es que podemos imprimir la cadena de respuesta formateada ❶ en lugar de escribirla en un archivo, porque la salida no es especialmente larga.

Everything in this program should look familiar, because we've used it all in the previous two chapters. The main difference here is that we can print the formatted response string ❶ instead of writing it to a file, because the output is not particularly long.

La salida es un diccionario de información sobre el artículo con el ID 31353677 :

The output is a dictionary of information about the article with the ID 31353677 :

```
{
  "by": "sohkamyung",
  ❶ "descendants": 302,
  "id": 31353677,
  ❷ "kids": [
    31354987,
    31354235,
    --snip--
  ],
  "score": 785,
  "time": 1652361401,
  ❸ "title": "Astronomers reveal first image of the black hole
    at the heart of our galaxy",
  "type": "story",
  ❹ "url": "https://public.nrao.edu/news/.../"
}
```

El diccionario contiene una serie de claves con las que podemos trabajar. La clave "descendants" nos indica el número de comentarios que ha recibido el artículo ❶ . La clave "kids" proporciona los ID de todos los comentarios realizados directamente en respuesta a este envío ❷ . Cada uno de estos comentarios puede tener también comentarios propios, por lo que el número de descendientes que tiene un envío suele ser mayor que su número de hijos. Podemos ver el título del artículo que se está discutiendo ❸ y también una URL del artículo que se está discutiendo ❹ .

The dictionary contains a number of keys we can work with. The key "descendants" tells us the number of comments the article has received ❶ . The key "kids" provides the IDs of all comments made directly in response to this submission ❷ . Each of these comments might have comments of their own as well, so the number of descendants a submission has is usually greater than its number of kids. We can see the title of the article being discussed ❸ and a URL for the article being discussed as well ❹ .

La siguiente URL devuelve una lista simple de todos los ID de los principales artículos actuales en Hacker News:

The following URL returns a simple list of all the IDs of the current top articles on Hacker News:

<https://hacker-news.firebaseio.com/v0/topstories.json>

Podemos utilizar esta llamada para averiguar qué artículos están en la página principal en este momento, y luego generar una serie de llamadas a la API similares a la que acabamos de examinar. Con este enfoque, podemos imprimir un resumen de todos los artículos que están en la portada de Hacker News en este momento:

We can use this call to find out which articles are on the home page right now, and then generate a series of API calls similar to the one we just examined. With this approach, we can print a summary of all the articles on the front page of Hacker News at the moment:

hn_submissions.py

hn_submissions.py

```
from operator import itemgetter

import requests

# Make an API call and check the response.
❶ url = "https://hacker-news.firebaseio.com/v0/topstories.json"
r = requests.get(url)
print(f"Status code: {r.status_code}")

# Process information about each submission.
❷ submission_ids = r.json()
❸ submission_dicts = []
for submission_id in submission_ids[:5]:
    # Make a new API call for each submission.
    ❹ url = f"https://hacker-news.firebaseio.com/v0/item/{submission_id}.json"
    r = requests.get(url)
    print(f"id: {submission_id}\tstatus: {r.status_code}")
    response_dict = r.json()

    # Build a dictionary for each article.
    ❺ submission_dict = {
        'title': response_dict['title'],
```

```

        'hn_link': f'https://news.ycombinator.com/item?id={submission_id}',
        'comments': response_dict['descendants'],
    }
    ❸ submission_dicts.append(submission_dict)

    ❹ submission_dicts = sorted(submission_dicts, key=itemgetter('comments'),
                               reverse=True)

    ❺ for submission_dict in submission_dicts:
        print(f"\nTitle: {submission_dict['title']}")
        print(f"Discussion link: {submission_dict['hn_link']}")
        print(f"Comments: {submission_dict['comments']}")

```

En primer lugar, hacemos una llamada a la API e imprimimos el estado de la respuesta ❶ . Esta llamada a la API devuelve una lista que contiene los ID de hasta 500 de los artículos más populares de Hacker News en el momento en que se realiza la llamada. A continuación, convertimos el objeto respuesta en una lista Python ❷ , que asignamos a `submission_ids` . Utilizaremos estos ID para construir un conjunto de diccionarios, cada uno de los cuales contiene información sobre uno de los envíos actuales.

First, we make an API call and print the status of the response ❶ . This API call returns a list containing the IDs of up to 500 of the most popular articles on Hacker News at the time the call is issued. We then convert the response object to a Python list ❷ , which we assign to `submission_ids` . We'll use these IDs to build a set of dictionaries, each of which contains information about one of the current submissions.

Creamos una lista vacía llamada `submission_dicts` para almacenar estos diccionarios ❸ . A continuación, hacemos un bucle con los identificadores de los 30 mejores envíos. Hacemos una nueva llamada a la API para cada envío generando una URL que incluya el valor actual de `submission_id` ❹ . Imprimimos el estado de cada solicitud junto con su ID, para saber si se ha realizado correctamente.

We set up an empty list called `submission_dicts` to store these dictionaries ❸ . We then loop through the IDs of the top 30 submissions. We make a new API call for each submission by generating a URL that includes the current value of `submission_id` ❹ . We print the status of each request along with its ID, so we can see whether it's successful.

A continuación, creamos un diccionario para el envío que se está procesando en ese momento ❺ . Almacenamos el título del envío, un enlace a la página de discusión de ese

artículo y el número de comentarios que ha recibido hasta el momento. A continuación, añadimos cada `submission_dict` a la lista `submission_dicts` ❹ .

Next, we create a dictionary for the submission currently being processed ❺ . We store the title of the submission, a link to the discussion page for that item, and the number of comments the article has received so far. Then we append each `submission_dict` to the list `submission_dicts` ❻ .

Cada envío en Hacker News se clasifica según una puntuación general basada en una serie de factores, entre ellos cuántas veces ha sido votado, cuántos comentarios ha recibido y lo reciente que es el envío. Queremos ordenar la lista de diccionarios por el número de comentarios. Para ello, utilizamos una función llamada `itemgetter()` ❼ , que procede del módulo `operator` . Le pasamos a esta función la clave `'comments'` , y extrae el valor asociado a esa clave de cada diccionario de la lista. A continuación, la función `sorted()` utiliza este valor como base para ordenar la lista. Ordenamos la lista en orden inverso, para colocar primero las historias más comentadas.

Each submission on Hacker News is ranked according to an overall score based on a number of factors, including how many times it's been voted on, how many comments it's received, and how recent the submission is. We want to sort the list of dictionaries by the number of comments. To do this, we use a function called `itemgetter()` ❼ , which comes from the `operator` module. We pass this function the key `'comments'` , and it pulls the value associated with that key from each dictionary in the list. The `sorted()` function then uses this value as its basis for sorting the list. We sort the list in reverse order, to place the most-commented stories first.

Una vez ordenada la lista, la recorreremos en bucle ❽ e imprimimos tres datos sobre cada uno de los envíos principales: el título, un enlace a la página de discusión y el número de comentarios que tiene actualmente el envío:

Once the list is sorted, we loop through the list ❽ and print out three pieces of information about each of the top submissions: the title, a link to the discussion page, and the number of comments the submission currently has:

Status code: 200

id: 31390506 status: 200

id: 31389893 status: 200

id: 31390742 status: 200

--snip--

Title: Fly.io: The reclaimer of Heroku's magic

Discussion link: <https://news.ycombinator.com/item?id=31390506>

Comments: 134

Title: The weird Hewlett Packard FreeDOS option

Discussion link: <https://news.ycombinator.com/item?id=31389893>

Comments: 64

Title: Modern JavaScript Tutorial

Discussion link: <https://news.ycombinator.com/item?id=31390742>

Comments: 20

--snip--

Utilizarías un proceso similar para acceder a la información y analizarla con cualquier API. Con estos datos, podrías hacer una visualización que mostrara qué envíos han inspirado los debates recientes más activos. Esta es también la base de las aplicaciones que proporcionan una experiencia de lectura personalizada para sitios como Hacker News. Para saber más sobre el tipo de información a la que puedes acceder a través de la API de Hacker News, visita la página de documentación en <https://github.com/HackerNews/API>.

You would use a similar process to access and analyze information with any API. With this data, you could make a visualization showing which submissions have inspired the most active recent discussions. This is also the basis for apps that provide a customized reading experience for sites like Hacker News. To learn more about what kind of information you can access through the Hacker News API, visit the documentation page at <https://github.com/HackerNews/API>.

Nota (Note)

En ocasiones, Hacker News permite que las empresas a las que apoya realicen publicaciones especiales sobre contratación, y los comentarios están desactivados en estas publicaciones. Si ejecutas este programa mientras uno de estos posts está presente, obtendrás un `KeyError`. Si esto causa un problema, puedes envolver el código que construye `submission_dict` en un bloque `try-except` y saltarte estos posts.

Hacker News sometimes allows companies it supports to make special hiring posts, and comments are disabled on these posts. If you run this program while one of these posts is present, you'll get a `KeyError`. If this causes an issue, you can wrap the code that builds `submission_dict` in a `try-except` block and skip over these posts.

Pruébalo tú mismo (Try It Yourself)

17-1. Otros idiomas: Modifica la llamada a la API en `python_repos.py` para que genere un gráfico que muestre los proyectos más populares en otros idiomas. Prueba con idiomas como *JavaScript*, *Ruby*, *C*, *Java*, *Perl*, *Haskell*, y *Go*.

17-1. Other Languages: Modify the API call in `python_repos.py` so it generates a chart showing the most popular projects in other languages. Try languages such as *JavaScript*, *Ruby*, *C*, *Java*, *Perl*, *Haskell*, and *Go*.

17-2. Discusiones activas: Utilizando los datos de `hn_submissions.py`, haz un gráfico de barras que muestre los debates más activos que se están produciendo actualmente en Hacker News. La altura de cada barra debe corresponder al número de comentarios que tiene cada envío. La etiqueta de cada barra debe incluir el título del envío y actuar como enlace a la página de discusión de ese envío. Si obtienes un `KeyError` al crear un gráfico, utiliza un bloque `try-except` para saltarte los envíos promocionales.

17-2. Active Discussions: Using the data from `hn_submissions.py`, make a bar chart showing the most active discussions currently happening on Hacker News. The height of each bar should correspond to the number of comments each submission has. The label for each bar should include the submission's title and act as a link to the discussion page for that submission. If you get a `KeyError` when creating a chart, use a `try-except` block to skip over the promotional posts.

17-3. Probando `python_repos.py`: En `python_repos.py`, imprimimos el valor de `status_code` para asegurarnos de que la llamada a la API se realizó correctamente. Escribe un programa llamado `test_python_repos.py` que utilice `pytest` para afirmar que el valor de `status_code` es 200. Piensa en otras afirmaciones que puedas hacer: por ejemplo, que el número de elementos devueltos sea el esperado y que el número total de repositorios sea

mayor que una cantidad determinada.

17-3. Testing `python_repos.py`: In `python_repos.py`, we printed the value of `status_code` to make sure the API call was successful. Write a program called `test_python_repos.py` that uses `pytest` to assert that the value of `status_code` is 200. Figure out some other assertions you can make: for example, that the number of items returned is expected and that the total number of repositories is greater than a certain amount.

17-4. Exploración adicional: Visita la documentación de Plotly y de la API de GitHub o de la API de Hacker News. Utiliza parte de la información que encuentres allí para personalizar el estilo de los gráficos que ya hemos hecho o para obtener información diferente y crear tus propias visualizaciones. Si tienes curiosidad por explorar otras API, echa un vistazo a las API mencionadas en el repositorio de GitHub en <https://github.com/public-apis>.

17-4. Further Exploration: Visit the documentation for Plotly and either the GitHub API or the Hacker News API. Use some of the information you find there to either customize the style of the plots we've already made or pull some different information and create your own visualizations. If you're curious about exploring other APIs, take a look at the APIs mentioned in the GitHub repository at <https://github.com/public-apis>.

Resumen (Summary)

En este capítulo has aprendido a utilizar las API para escribir programas autónomos que recopilan automáticamente los datos que necesitan y los utilizan para crear una visualización. Utilizaste la API de GitHub para explorar los proyectos Python con más estrellas en GitHub, y también echaste un breve vistazo a la API de Hacker News. Aprendiste a utilizar el paquete `Peticiones` para realizar automáticamente una llamada a la API y a procesar los resultados de esa llamada. También introdujimos algunos ajustes de Plotly que personalizan aún más el aspecto de los gráficos que generas.

In this chapter, you learned how to use APIs to write self-contained programs that automatically gather the data they need and use that data to create a visualization. You used the GitHub API to explore the most-starred Python projects on GitHub, and you also looked briefly at the Hacker News API. You learned how to use the `Requests` package to automatically issue an API call and how to process the results of that call. We also

introduced some Plotly settings that further customize the appearance of the charts you generate.

En el próximo capítulo, utilizarás Django para construir una aplicación web como proyecto final.

In the next chapter, you'll use Django to build a web application as your final project.

Capítulo 18: Primeros pasos con Django (Chapter 18: Getting Started with Django)

18

Primeros pasos con Django (18 Getting Started with Django)



DESCRIPCIÓN DE LA
IMAGEN (En español):

La imagen muestra un dibujo estilizado de una serpiente enrollada alrededor de una hélice de avión. La serpiente parece animada y tiene una expresión audaz en el rostro, con ojos grandes y llamativos. La hélice está en el centro, con una forma cónica y detalles que sugieren movimiento. El fondo es de color gris claro, lo que resalta el arte en blanco y negro del reptil y la hélice.

A medida que Internet ha evolucionado, la línea que separa los sitios web de las aplicaciones móviles se ha difuminado. Tanto los sitios web como las aplicaciones ayudan a los usuarios a interactuar con los datos de diversas formas. Afortunadamente, puedes utilizar Django para construir un único proyecto que sirva tanto para un sitio web dinámico como para un conjunto de aplicaciones móviles. *Django* es el más popular de Python *web framework*, un conjunto de herramientas diseñadas para construir aplicaciones web interactivas. En este capítulo, aprenderás a utilizar Django para construir un proyecto llamado Learning Log, un sistema de diario online que te permite llevar un registro de la información que has aprendido sobre diferentes temas.

As the internet has evolved, the line between websites and mobile apps has blurred. Websites and apps both help users interact with data in a variety of ways. Fortunately, you can use Django to build a single project that serves a dynamic website as well as a set of mobile apps. *Django* is Python's most popular *web framework*, a set of tools designed for

building interactive web applications. In this chapter, you'll learn how to use Django to build a project called Learning Log, an online journal system that lets you keep track of information you've learned about different topics.

Escribiremos una especificación para este proyecto y, a continuación, definiremos modelos para los datos con los que trabajará la aplicación. Utilizaremos el sistema de administración de Django para introducir algunos datos iniciales, y luego escribiremos vistas y plantillas para que Django pueda construir las páginas del sitio.

We'll write a specification for this project, and then define models for the data the app will work with. We'll use Django's admin system to enter some initial data, and then write views and templates so Django can build the site's pages.

Django puede responder a peticiones de páginas y facilitar la lectura y escritura en una base de datos, gestionar usuarios y mucho más. En los Capítulos 19 y 20 , perfeccionarás el proyecto Learning Log, y luego lo desplegarás en un servidor en vivo para que tú (y todos los demás en el mundo) podáis utilizarlo.

Django can respond to page requests and make it easier to read and write to a database, manage users, and much more. In Chapters 19 and 20 , you'll refine the Learning Log project, and then deploy it to a live server so you (and everyone else in the world) can use it.

Configurar un proyecto (*Setting Up a Project*)

Cuando empiezas a trabajar en algo tan importante como una aplicación web, primero tienes que describir los objetivos del proyecto en una especificación, o *spec* . Una vez que tengas un conjunto claro de objetivos, puedes empezar a identificar tareas manejables para alcanzar esos objetivos.

When starting work on something as significant as a web app, you first need to describe the project's goals in a specification, or *spec* . Once you have a clear set of goals, you can start to identify manageable tasks to achieve those goals.

En esta sección, escribiremos una especificación para Learning Log y empezaremos a trabajar en la primera fase del proyecto. Esto implicará configurar un entorno virtual y construir los aspectos iniciales de un proyecto Django.

In this section, we'll write a spec for Learning Log and start working on the first phase of the project. This will involve setting up a virtual environment and building out the initial aspects of a Django project.

Escribir una especificación (*Writing a Spec*)

Una especificación completa detalla los objetivos del proyecto, describe su funcionalidad y habla de su aspecto e interfaz de usuario. Como cualquier buen proyecto o plan de empresa, una especificación debe mantenerte centrado y ayudarte a mantener tu proyecto en el buen camino. No escribiremos aquí una especificación completa del proyecto, pero estableceremos algunos objetivos claros para mantener centrado el proceso de desarrollo. Estas son las especificaciones que utilizaremos:

A full spec details the project goals, describes the project's functionality, and discusses its appearance and user interface. Like any good project or business plan, a spec should keep you focused and help keep your project on track. We won't write a full project spec here, but we'll lay out a few clear goals to keep the development process focused. Here's the spec we'll use:

Escribiremos una aplicación web llamada Learning Log que permita a los usuarios registrar los temas que les interesan y hacer anotaciones en un diario a medida que aprenden sobre cada tema. La página de inicio de Learning Log describirá el sitio e invitará a los usuarios a registrarse o a iniciar sesión. Una vez conectado, un usuario puede crear nuevos temas, añadir nuevas entradas y leer y editar las entradas existentes.

We'll write a web app called Learning Log that allows users to log the topics they're interested in and make journal entries as they learn about each topic. The Learning Log home page will describe the site and invite users to either register or log in. Once logged in, a user can create new topics, add new entries, and read and edit existing entries.

Cuando investigas un tema nuevo, llevar un diario de lo que has aprendido puede ayudarte a hacer un seguimiento de la información nueva y de la que ya has encontrado. Esto es

especialmente cierto cuando se estudian temas técnicos. Una buena aplicación, como la que vamos a crear, puede ayudar a que este proceso sea más eficaz.

When you're researching a new topic, maintaining a journal of what you've learned can help you keep track of new information and information you've already found. This is especially true when studying technical subjects. A good app, like the one we'll be creating, can help make this process more efficient.

Crear un entorno virtual (*Creating a Virtual Environment*)

Para trabajar con Django, primero configuraremos un entorno virtual. Un *virtual environment* es un lugar de tu sistema donde puedes instalar paquetes y aislarlos de todos los demás paquetes de Python. Separar las bibliotecas de un proyecto de las de otros proyectos es beneficioso y será necesario cuando despluguemos Learning Log en un servidor en el Capítulo 20 .

To work with Django, we'll first set up a virtual environment. A *virtual environment* is a place on your system where you can install packages and isolate them from all other Python packages. Separating one project's libraries from other projects is beneficial and will be necessary when we deploy Learning Log to a server in Chapter 20 .

Crea un nuevo directorio para tu proyecto llamado *learning_log* , cambia a ese directorio en un terminal, e introduce el siguiente código para crear un entorno virtual:

Create a new directory for your project called *learning_log* , switch to that directory in a terminal, and enter the following code to create a virtual environment:

```
learning_log$ python -m venv ll_env
learning_log$
```

Aquí estamos ejecutando el módulo de entorno virtual `venv` y utilizándolo para crear un entorno llamado `ll_env` (ten en cuenta que este nombre empieza por dos s minúsculas `L` , no dos unos). Si utilizas un comando como `python3` al ejecutar programas o instalar paquetes, asegúrate de utilizar ese comando aquí.

Here we're running the `venv` virtual environment module and using it to create an environment named `ll_env` (note that this name starts with two lowercase *L*s, not two ones). If you use a command such as `python3` when running programs or installing packages, make sure to use that command here.

Activar el Entorno Virtual (*Activating the Virtual Environment*)

Ahora tenemos que activar el entorno virtual, utilizando el siguiente comando:

Now we need to activate the virtual environment, using the following command:

```
learning_log$ source ll_env/bin/activate
(ll_env)learning_log$
```

Este comando ejecuta el script *activate* en `ll_env/bin/`. Cuando el entorno esté activo, verás el nombre del entorno entre paréntesis. Esto indica que puedes instalar nuevos paquetes en el entorno y utilizar paquetes ya instalados. Los paquetes que instales en `ll_env` no estarán disponibles cuando el entorno esté inactivo.

This command runs the script *activate* in `ll_env/bin/`. When the environment is active, you'll see the name of the environment in parentheses. This indicates that you can install new packages to the environment and use packages that have already been installed. Packages you install in `ll_env` will not be available when the environment is inactive.

Nota (*Note*)

Si utilizas Windows, utiliza el comando `ll_env\Scripts\activate` (sin la palabra `source`) para activar el entorno virtual. Si utilizas PowerShell, es posible que tengas que escribir `Activate` en mayúsculas.

If you're using Windows, use the command `ll_env\Scripts\activate` (without the word `source`) to activate the virtual environment. If you're using PowerShell, you might need to capitalize `Activate`.

Para dejar de utilizar un entorno virtual, introduce deactivate :

To stop using a virtual environment, enter deactivate :

```
(ll_env)learning_log$ deactivate
learning_log$
```

El entorno también quedará inactivo cuando cierres el terminal en el que se está ejecutando.

The environment will also become inactive when you close the terminal it's running in.

Instalar Django (*Installing Django*)

Con el entorno virtual activado, introduce lo siguiente para actualizar pip e instalar Django:

With the virtual environment activated, enter the following to update pip and install Django:

```
(ll_env)learning_log$ pip install --upgrade pip
(ll_env)learning_log$ pip install django
Collecting django
--snip--
Installing collected packages: sqlparse, asgiref, django
Successfully installed asgiref-3.5.2 django-4.1 sqlparse-0.4.2
(ll_env)learning_log$
```

Dado que descarga recursos de diversas fuentes, pip se actualiza con bastante frecuencia. Es una buena idea actualizar pip siempre que crees un nuevo entorno virtual.

Because it downloads resources from a variety of sources, pip is upgraded fairly often. It's a good idea to upgrade pip whenever you make a new virtual environment.

Ahora estamos trabajando en un entorno virtual, por lo que el comando para instalar Django es el mismo en todos los sistemas. No es necesario utilizar comandos más largos, como `python -m pip install package_name`, ni incluir la bandera `--user`. Ten en cuenta que Django sólo estará disponible cuando el entorno `ll_env` esté activo.

We're working in a virtual environment now, so the command to install Django is the same on all systems. There's no need to use longer commands, such as `python -m pip install package_name`, or to include the `--user` flag. Keep in mind that Django will be available only when the `ll_env` environment is active.

Nota (Note)

Django lanza una nueva versión aproximadamente cada ocho meses, por lo que es posible que veas una versión más reciente cuando instales Django. Lo más probable es que este proyecto funcione tal y como está escrito aquí, incluso en versiones más recientes de Django. Si quieres asegurarte de utilizar la misma versión de Django que ves aquí, utiliza el comando `pip install django==4.1.*`. Esto instalará la última versión de Django 4.1. Si tienes algún problema relacionado con la versión que estás utilizando, consulta los recursos en línea de este libro en https://ehmatthes.github.io/pcc_3e.

Django releases a new version about every eight months, so you may see a newer version when you install Django. This project will most likely work as it's written here, even on newer versions of Django. If you want to make sure to use the same version of Django you see here, use the command `pip install django==4.1.*`. This will install the latest release of Django 4.1. If you have any issues related to the version you're using, see the online resources for this book at https://ehmatthes.github.io/pcc_3e.

Crear un proyecto en Django (Creating a Project in Django)

Sin salir del entorno virtual activo (recuerda buscar `ll_env` entre paréntesis en el prompt del terminal), introduce los siguientes comandos para crear un nuevo proyecto:

Without leaving the active virtual environment (remember to look for *ll_env* in parentheses in the terminal prompt), enter the following commands to create a new project:

❶ (ll_env)learning_log\$ django-admin startproject ll_project .

❷ (ll_env)learning_log\$ ls

ll_env ll_project manage.py

❸ (ll_env)learning_log\$ ls ll_project

__init__.py asgi.py settings.py urls.py wsgi.py

El comando `startproject` ❶ indica a Django que cree un nuevo proyecto llamado *ll_project* . El punto (`.`) al final del comando crea el nuevo proyecto con una estructura de directorios que facilitará el despliegue de la aplicación en un servidor cuando hayamos terminado de desarrollarla.

The `startproject` command ❶ tells Django to set up a new project called *ll_project* . The dot (`.`) at the end of the command creates the new project with a directory structure that will make it easy to deploy the app to a server when we're finished developing it.

Nota (Note)

No olvides este punto, o podrías encontrarte con algunos problemas de configuración cuando despliegues la aplicación. Si olvidas el punto, borra los archivos y carpetas creados (excepto *ll_env*) y vuelve a ejecutar el comando.

Don't forget this dot, or you might run into some configuration issues when you deploy the app. If you forget the dot, delete the files and folders that were created (except *ll_env*) and run the command again.

Si ejecutas el comando `ls` (`dir` en Windows) ❷ verás que Django ha creado un nuevo directorio llamado *ll_project* . También ha creado un archivo *manage.py* , que es un pequeño programa que recibe comandos y los envía a la parte relevante de Django. Utilizaremos estos comandos para gestionar tareas, como trabajar con bases de datos y ejecutar servidores.

Running the `ls` command (`dir` on Windows) ❷ shows that Django has created a new directory called `ll_project`. It also created a `manage.py` file, which is a short program that takes in commands and feeds them to the relevant part of Django. We'll use these commands to manage tasks, such as working with databases and running servers.

El directorio `ll_project` contiene cuatro archivos ❸ ; los más importantes son `settings.py`, `urls.py` y `wsgi.py`. El archivo `settings.py` controla la forma en que Django interactúa con tu sistema y gestiona tu proyecto. Modificaremos algunos de estos ajustes y añadiremos algunos ajustes propios a medida que evolucione el proyecto. El archivo `urls.py` indica a Django qué páginas debe construir en respuesta a las peticiones del navegador. El archivo `wsgi.py` ayuda a Django a servir los archivos que crea. El nombre del archivo es un acrónimo de "web server gateway interface".

The `ll_project` directory contains four files ❸ ; the most important are `settings.py`, `urls.py`, and `wsgi.py`. The `settings.py` file controls how Django interacts with your system and manages your project. We'll modify a few of these settings and add some settings of our own as the project evolves. The `urls.py` file tells Django which pages to build in response to browser requests. The `wsgi.py` file helps Django serve the files it creates. The filename is an acronym for "web server gateway interface."

Crear la base de datos (Creating the Database)

Django almacena la mayor parte de la información de un proyecto en una base de datos, así que lo siguiente que tenemos que hacer es crear una base de datos con la que Django pueda trabajar. Introduce el siguiente comando (aún en un entorno activo):

Django stores most of the information for a project in a database, so next we need to create a database that Django can work with. Enter the following command (still in an active environment):

```
(ll_env)learning_log$ python manage.py migrate
```

❶ Operations to perform:

Apply all migrations: admin, auth, contenttypes, sessions

Running migrations:

Applying contenttypes.0001_initial... OK

Applying auth.0001_initial... OK

--snip--

Applying sessions.0001_initial... OK


```
❷ (ll_env)learning_log$ ls
db.sqlite3 ll_env ll_project manage.py
```

Cada vez que modificamos una base de datos, decimos que estamos *migrating* la base de datos. Ejecutar el comando `migrate` por primera vez le dice a Django que se asegure de que la base de datos coincide con el estado actual del proyecto. La primera vez que ejecutemos este comando en un nuevo proyecto que utilice SQLite (más sobre SQLite en un momento), Django creará una nueva base de datos para nosotros. Aquí, Django informa de que preparará la base de datos para almacenar la información que necesita para gestionar las tareas administrativas y de autenticación ❶ .

Anytime we modify a database, we say we're *migrating* the database. Issuing the `migrate` command for the first time tells Django to make sure the database matches the current state of the project. The first time we run this command in a new project using SQLite (more about SQLite in a moment), Django will create a new database for us. Here, Django reports that it will prepare the database to store information it needs to handle administrative and authentication tasks ❶ .

Al ejecutar el comando `ls` se muestra que Django ha creado otro archivo llamado `db.sqlite3` ❷ . *SQLite* es una base de datos que se ejecuta desde un único archivo; es ideal para escribir aplicaciones sencillas porque no tendrás que prestar mucha atención a la gestión de la base de datos.

Running the `ls` command shows that Django created another file called `db.sqlite3` ❷ . *SQLite* is a database that runs off a single file; it's ideal for writing simple apps because you won't have to pay much attention to managing the database.

Nota (Note)

En un entorno virtual activo, utiliza el comando `python` para ejecutar los comandos `manage.py` , aunque utilices algo diferente, como `python3` , para ejecutar otros programas. En un entorno virtual, el comando `python` hace referencia a la versión de Python que se utilizó para crear el entorno virtual.

In an active virtual environment, use the command `python` to run `manage.py` commands, even if you use something different, like `python3` , to run other programs. In a virtual

environment, the command `python` refers to the version of Python that was used to create the virtual environment.

Ver el proyecto (Viewing the Project)

Asegurémonos de que Django ha configurado el proyecto correctamente. Introduce el comando `runserver` para ver el proyecto en su estado actual:

Let's make sure that Django has set up the project properly. Enter the `runserver` command to view the project in its current state:

```
(ll_env)learning_log$ python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...
```

- ❶ System check identified no issues (0 silenced).
 - May 19, 2022 - 21:52:35
 - ❷ Django version 4.1, using settings 'll_project.settings'
 - ❸ Starting development server at `http://127.0.0.1:8000/`
- Quit the server with `CONTROL-C`.

Django debería iniciar un servidor llamado *development server*, para que puedas ver el proyecto en tu sistema y comprobar si funciona correctamente. Cuando solicitas una página introduciendo una URL en un navegador, el servidor Django responde a esa solicitud construyendo la página apropiada y enviándola al navegador.

Django should start a server called the *development server*, so you can view the project on your system to see how well it works. When you request a page by entering a URL in a browser, the Django server responds to that request by building the appropriate page and sending it to the browser.

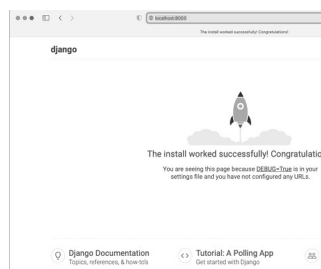
En primer lugar, Django comprueba que el proyecto está configurado correctamente ❶ ; a continuación, informa de la versión de Django en uso y del nombre del archivo de configuración en uso ❷ . Por último, informa de la URL en la que se sirve el proyecto ❸ . La URL `http://127.0.0.1:8000/` indica que el proyecto está a la escucha de peticiones en el puerto 8000 de tu ordenador, que se denomina *localhost*. El término *localhost* se refiere a

un servidor que sólo procesa peticiones en tu sistema; no permite que nadie más vea las páginas que estás desarrollando.

Django first checks to make sure the project is set up properly ❶ ; it then reports the version of Django in use and the name of the settings file in use ❷ . Finally, it reports the URL where the project is being served ❸ . The URL `http://127.0.0.1:8000/` indicates that the project is listening for requests on port 8000 on your computer, which is called a localhost. The term *localhost* refers to a server that only processes requests on your system; it doesn't allow anyone else to see the pages you're developing.

Abre un navegador web e introduce la URL `http://localhost:8000/` , o `http://127.0.0.1:8000/` si la primera no funciona. Deberías ver algo como la [Figura 18-1](#) : una página que Django crea para informarte de que todo funciona correctamente hasta el momento. Mantén el servidor en funcionamiento por ahora, pero cuando quieras detenerlo, pulsa CTRL-C en el terminal donde se emitió el comando `runserver` .

Open a web browser and enter the URL `http://localhost:8000/` , or `http://127.0.0.1:8000/` if the first one doesn't work. You should see something like [Figure 18-1](#) : a page that Django creates to let you know everything is working properly so far. Keep the server running for now, but when you want to stop the server, press CTRL-C in the terminal where the `runserver` command was issued.



DESCRIPCIÓN DE LA IMAGEN (En español):

django La instalación ha funcionado con éxito. ¡Felicitaciones! Ver notas de la versión para el desarrollo de Django. Usted está viendo esta página porque `DEBUG=True` está en su archivo de configuración y no ha configurado ninguna URL. Documentación de Django: Temas, referencias y guías. Tutorial: Una aplicación de encuestas. Comience

con Django. Comunidad
de Django: Encuentra
ayuda o contribuye.

Figura 18-1 : Hasta ahora todo funciona correctamente.

Figure 18-1 : Everything is working so far.

Nota (Note)

Si recibes el mensaje de error "Ese puerto ya está en uso", dile a Django que utilice un puerto diferente introduciendo `python manage.py runserver 8001` y luego pasando por números superiores hasta que encuentres un puerto abierto.

If you receive the error message "That port is already in use," tell Django to use a different port by entering `python manage.py runserver 8001` and then cycling through higher numbers until you find an open port.

Pruébalo tú mismo (Try It Yourself)

18-1. Nuevos proyectos: Para hacerte una mejor idea de lo que hace Django, crea un par de proyectos vacíos y observa lo que crea Django. Crea una nueva carpeta con un nombre sencillo, como *tik_gram* o *insta_tok* (fuera de tu directorio *learning_log*), navega hasta esa carpeta en un terminal y crea un entorno virtual. Instala Django y ejecuta el comando `django-admin.py startproject tg_project .` (asegurándote de incluir el punto al final del comando).

18-1. New Projects: To get a better idea of what Django does, build a couple empty projects and look at what Django creates. Make a new folder with a simple name, like *tik_gram* or *insta_tok* (outside of your *learning_log* directory), navigate to that folder in a terminal, and create a virtual environment. Install Django and run the command `django-admin.py startproject tg_project .` (making sure to include the dot at the end of the command).

Observa los archivos y carpetas que crea este comando, y compáralos con el Registro de Aprendizaje. Hazlo varias veces, hasta que te familiarices con lo que crea Django al iniciar

un nuevo proyecto. Después, borra los directorios del proyecto si lo deseas.

Look at the files and folders this command creates, and compare them to Learning Log. Do this a few times, until you're familiar with what Django creates when starting a new project. Then delete the project directories if you wish.

Iniciar una aplicación (*Starting an App*)

Un proyecto Django *project* está organizado como un grupo de *apps* individuales que trabajan juntos para hacer que el proyecto funcione como un todo. Por ahora, crearemos una app para que realice la mayor parte del trabajo de nuestro proyecto. Añadiremos otra app en el Capítulo 19 para gestionar las cuentas de usuario.

A Django *project* is organized as a group of individual *apps* that work together to make the project work as a whole. For now, we'll create one app to do most of our project's work. We'll add another app in Chapter 19 to manage user accounts.

Deberías dejar el servidor de desarrollo funcionando en la ventana de terminal que abriste antes. Abre una nueva ventana de terminal (o pestaña) y navega hasta el directorio que contiene *manage.py*. Activa el entorno virtual y, a continuación, ejecuta el comando `startapp` :

You should leave the development server running in the terminal window you opened earlier. Open a new terminal window (or tab) and navigate to the directory that contains *manage.py*. Activate the virtual environment, and then run the `startapp` command:

```
learning_log$ source ll_env/bin/activate
(ll_env)learning_log$ python manage.py startapp learning_logs
❶ (ll_env)learning_log$ ls
db.sqlite3 learning_logs ll_env ll_project manage.py
❷ (ll_env)learning_log$ ls learning_logs/
__init__.py admin.py apps.py migrations models.py tests.py views.py
```

El comando `startapp appname` indica a Django que cree la infraestructura necesaria para crear una aplicación. Cuando mires ahora en el directorio del proyecto, verás una nueva

carpeta llamada *learning_logs* ❶ . Utiliza el comando `ls` para ver lo que Django ha creado ❷ . Los archivos más importantes son *models.py* , *admin.py* , y *views.py* . Utilizaremos *models.py* para definir los datos que queremos gestionar en nuestra app. Veremos *admin.py* y *views.py* un poco más adelante.

The command `startapp appname` tells Django to create the infrastructure needed to build an app. When you look in the project directory now, you'll see a new folder called *learning_logs* ❶ . Use the `ls` command to see what Django has created ❷ . The most important files are *models.py* , *admin.py* , and *views.py* . We'll use *models.py* to define the data we want to manage in our app. We'll look at *admin.py* and *views.py* a little later.

Definir modelos (*Defining Models*)

Pensemos un momento en nuestros datos. Cada usuario tendrá que crear una serie de temas en su registro de aprendizaje. Cada entrada que hagan estará vinculada a un tema, y estas entradas se mostrarán como texto. También necesitaremos almacenar la marca de tiempo de cada entrada para poder mostrar a los usuarios cuándo hicieron cada una.

Let's think about our data for a moment. Each user will need to create a number of topics in their learning log. Each entry they make will be tied to a topic, and these entries will be displayed as text. We'll also need to store the timestamp of each entry so we can show users when they made each one.

Abre el archivo *models.py* y mira el contenido existente:

Open the file *models.py* and look at its existing content:

models.py

models.py

```
from django.db import models
```

```
# Create your models here.
```

Se está importando un módulo llamado `models` , y se nos invita a crear modelos propios. Un *model* indica a Django cómo trabajar con los datos que se almacenarán en la aplicación. Un modelo es una clase; tiene atributos y métodos, como todas las clases de las que hemos hablado. Este es el modelo para los temas que almacenarán los usuarios:

A module called `models` is being imported, and we're being invited to create models of our own. A *model* tells Django how to work with the data that will be stored in the app. A model is a class; it has attributes and methods, just like every class we've discussed. Here's the model for the topics users will store:

```
from django.db import models

class Topic(models.Model):
    """A topic the user is learning about."""
    ❶ text = models.CharField(max_length=200)
    ❷ date_added = models.DateTimeField(auto_now_add=True)

    ❸ def __str__(self):
        """Return a string representation of the model."""
        return self.text
```

Hemos creado una clase llamada `Topic` , que hereda de `Model` -una clase padre incluida en Django que define la funcionalidad básica de un modelo. Añadimos dos atributos a la clase `Topic` : `text` y `date_added` .

We've created a class called `Topic` , which inherits from `Model` —a parent class included in Django that defines a model's basic functionality. We add two attributes to the `Topic` class: `text` and `date_added` .

El atributo `text` es un `CharField` , un dato formado por caracteres o texto ❶ . Utilizarás `CharField` cuando quieras almacenar una pequeña cantidad de texto, como un nombre, un título o una ciudad. Cuando definimos un atributo `CharField` , tenemos que decirle a Django cuánto espacio debe reservar en la base de datos. Aquí le damos un `max_length` de 200 caracteres, que debería ser suficiente para contener la mayoría de los nombres de temas.

The `text` attribute is a `CharField`, a piece of data that's made up of characters or text ❶. You use `CharField` when you want to store a small amount of text, such as a name, a title, or a city. When we define a `CharField` attribute, we have to tell Django how much space it should reserve in the database. Here we give it a `max_length` of 200 characters, which should be enough to hold most topic names.

El atributo `date_added` es un `DateTimeField`, un dato que registrará una fecha y una hora ❷. Pasamos el argumento `auto_now_add=True`, que indica a Django que establezca automáticamente este atributo con la fecha y hora actuales cada vez que el usuario cree un nuevo tema.

The `date_added` attribute is a `DateTimeField`, a piece of data that will record a date and time ❷. We pass the argument `auto_now_add=True`, which tells Django to automatically set this attribute to the current date and time whenever the user creates a new topic.

Es una buena idea decirle a Django cómo quieres que represente una instancia de un modelo. Si un modelo tiene un método `__str__()`, Django llama a ese método cada vez que necesita generar una salida que haga referencia a una instancia de ese modelo. Aquí hemos escrito un método `__str__()` que devuelve el valor asignado al atributo `text` ❸.

It's a good idea to tell Django how you want it to represent an instance of a model. If a model has a `__str__()` method, Django calls that method whenever it needs to generate output referring to an instance of that model. Here we've written a `__str__()` method that returns the value assigned to the `text` attribute ❸.

Para ver los distintos tipos de campos que puedes utilizar en un modelo, consulta la página "Referencia de campos de modelo" en <https://docs.djangoproject.com/en/4.1/ref/models/fields>. No necesitarás toda la información ahora mismo, pero te será muy útil cuando desarrolles tus propios proyectos Django.

To see the different kinds of fields you can use in a model, see the "Model Field Reference" page at <https://docs.djangoproject.com/en/4.1/ref/models/fields>. You won't need all the information right now, but it will be extremely useful when you're developing your own Django projects.

Activar modelos (*Activating Models*)

Para utilizar nuestros modelos, tenemos que decirle a Django que incluya nuestra aplicación en el proyecto general. Abre *settings.py* (en el directorio *ll_project*); verás una sección que indica a Django qué aplicaciones están instaladas en el proyecto:

To use our models, we have to tell Django to include our app in the overall project. Open *settings.py* (in the *ll_project* directory); you'll see a section that tells Django which apps are installed in the project:

settings.py

settings.py

```
--snip--
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
--snip--
```

Añade nuestra aplicación a esta lista modificando `INSTALLED_APPS` para que tenga este aspecto:

Add our app to this list by modifying `INSTALLED_APPS` so it looks like this:

```
--snip--
INSTALLED_APPS = [
    # My apps.
    'learning_logs',
```

```
# Default django apps.  
'django.contrib.admin',  
--snip--  
]  
--snip--
```

Agrupar aplicaciones en un proyecto ayuda a seguirlas cuando el proyecto crece e incluye más aplicaciones. Aquí empezamos una sección llamada `My apps` , que por ahora sólo incluye `'learning_logs'` . Es importante que coloques tus propias aplicaciones antes de las aplicaciones por defecto, por si necesitas sustituir algún comportamiento de las aplicaciones por defecto por tu propio comportamiento personalizado.

Grouping apps together in a project helps keep track of them as the project grows to include more apps. Here we start a section called `My apps` , which includes only `'learning_logs'` for now. It's important to place your own apps before the default apps, in case you need to override any behavior of the default apps with your own custom behavior.

A continuación, tenemos que decirle a Django que modifique la base de datos para que pueda almacenar información relacionada con el modelo `Topic` . Desde el terminal, ejecuta el siguiente comando:

Next, we need to tell Django to modify the database so it can store information related to the model `Topic` . From the terminal, run the following command:

```
(ll_env)learning_log$ python manage.py makemigrations learning_logs  
Migrations for 'learning_logs':  
  learning_logs/migrations/0001_initial.py  
    - Create model Topic  
(ll_env)learning_log$
```

El comando `makemigrations` indica a Django que averigüe cómo modificar la base de datos para que pueda almacenar los datos asociados a los nuevos modelos que hayamos definido. La salida aquí muestra que Django ha creado un archivo de migración llamado `0001_initial.py` . Esta migración creará una tabla para el modelo `Topic` en la base de datos.

The command `makemigrations` tells Django to figure out how to modify the database so it can store the data associated with any new models we've defined. The output here shows that Django has created a migration file called `0001_initial.py`. This migration will create a table for the model `Topic` in the database.

Ahora aplicaremos esta migración y haremos que Django modifique la base de datos por nosotros:

Now we'll apply this migration and have Django modify the database for us:

```
(ll_env)learning_log$ python manage.py migrate
```

Operations to perform:

Apply all migrations: admin, auth, contenttypes, learning_logs, sessions

Running migrations:

Applying learning_logs.0001_initial... OK

La mayor parte de la salida de este comando es idéntica a la salida de la primera vez que ejecutamos el comando `migrate`. Tenemos que comprobar la última línea de esta salida, donde Django confirma que la migración para `learning_logs` funcionó OK.

Most of the output from this command is identical to the output from the first time we issued the `migrate` command. We need to check the last line in this output, where Django confirms that the migration for `learning_logs` worked OK.

Siempre que queramos modificar los datos que gestiona Learning Log, seguiremos estos tres pasos: modificar `models.py`, llamar a `makemigrations` en `learning_logs`, y decirle a Django que `migrate` el proyecto.

Whenever we want to modify the data that Learning Log manages, we'll follow these three steps: modify `models.py`, call `makemigrations` on `learning_logs`, and tell Django to `migrate` the project.

El sitio de administración de Django (*The Django Admin Site*)

Django facilita el trabajo con tus modelos a través de su sitio de administración. El sitio *admin site* de Django sólo está pensado para ser utilizado por los administradores del sitio; no está pensado para usuarios normales. En esta sección, configuraremos el sitio de administración y lo utilizaremos para añadir algunos temas a través del modelo `Topic`.

Django makes it easy to work with your models through its admin site. Django's *admin site* is only meant to be used by the site's administrators; it's not meant for regular users. In this section, we'll set up the admin site and use it to add some topics through the `Topic` model.

Configurar un superusuario (*Setting Up a Superuser*)

Django te permite crear un *superuser*, un usuario que tiene todos los privilegios disponibles en el sitio. Los *privileges* de un usuario controlan las acciones que puede realizar. La configuración de privilegios más restrictiva permite a un usuario leer únicamente la información pública del sitio. Los usuarios registrados suelen tener el privilegio de leer sus propios datos privados y alguna información seleccionada disponible sólo para los miembros. Para administrar eficazmente un proyecto, el propietario del sitio suele necesitar acceso a toda la información almacenada en él. Un buen administrador es cuidadoso con la información sensible de sus usuarios, porque éstos depositan mucha confianza en las aplicaciones a las que acceden.

Django allows you to create a *superuser*, a user who has all privileges available on the site. A user's *privileges* control the actions they can take. The most restrictive privilege settings allow a user to only read public information on the site. Registered users typically have the privilege of reading their own private data and some selected information available only to members. To effectively administer a project, the site owner usually needs access to all information stored on the site. A good administrator is careful with their users' sensitive information, because users put a lot of trust into the apps they access.

Para crear un superusuario en Django, introduce el siguiente comando y responde a las indicaciones:

To create a superuser in Django, enter the following command and respond to the prompts:

```
(ll_env)learning_log$ python manage.py createsuperuser
```

❶ Username (leave blank to use 'eric'): ll_admin

❷ Email address:

❸ Password:

Password (again):

Superuser created successfully.

```
(ll_env)learning_log$
```

Cuando emitas el comando `createsuperuser`, Django te pedirá que introduzcas un nombre de usuario para el superusuario ❶. Aquí estoy utilizando `ll_admin`, pero puedes introducir cualquier nombre de usuario que desees. Puedes introducir una dirección de correo electrónico o simplemente dejar este campo en blanco ❷. Tendrás que introducir tu contraseña dos veces ❸.

When you issue the command `createsuperuser`, Django prompts you to enter a username for the superuser ❶. Here I'm using `ll_admin`, but you can enter any username you want. You can enter an email address or just leave this field blank ❷. You'll need to enter your password twice ❸.

Nota (Note)

Cierta información sensible puede ocultarse a los administradores de un sitio. Por ejemplo, Django no almacena la contraseña que introduces; en su lugar, almacena una cadena derivada de la contraseña, denominada *hash*. Cada vez que introduces tu contraseña, Django realiza un hash de tu entrada y la compara con el hash almacenado. Si los dos hashes coinciden, estás autenticado. Al exigir que los hashes coincidan, Django se asegura de que si un atacante accede a la base de datos de un sitio, podrá leer los hashes almacenados, pero no las contraseñas. Cuando un sitio está configurado correctamente, es casi imposible obtener las contraseñas originales a partir de los hashes.

Some sensitive information can be hidden from a site's administrators. For example, Django doesn't store the password you enter; instead, it stores a string derived from the password, called a *hash*. Each time you enter your password, Django hashes your entry and compares it to the stored hash. If the two hashes match, you're authenticated. By requiring hashes to match, Django ensures that if an attacker gains access to a site's database, they'll be able to read the stored hashes but not the passwords. When a site is set up properly, it's almost impossible to get the original passwords from the hashes.

Registrar un modelo en el sitio de administración (Registering a Model with the Admin Site)

Django incluye algunos modelos en el sitio de administración de forma automática, como `User` y `Group`, pero los modelos que creamos deben añadirse manualmente.

Django includes some models in the admin site automatically, such as `User` and `Group`, but the models we create need to be added manually.

Cuando iniciamos la aplicación `learning_logs`, Django creó un archivo `admin.py` en el mismo directorio que `models.py`. Abre el archivo `admin.py`:

When we started the `learning_logs` app, Django created an `admin.py` file in the same directory as `models.py`. Open the `admin.py` file:

`admin.py`

`admin.py`

```
from django.contrib import admin
```

```
# Register your models here.
```

Para registrar `Topic` en el sitio admin, introduce lo siguiente:

To register `Topic` with the admin site, enter the following:

```
from django.contrib import admin
```

```
from .models import Topic
```

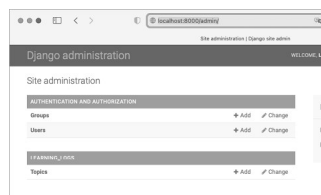
```
admin.site.register(Topic)
```

Este código importa primero el modelo que queremos registrar, `Topic` . El punto delante de `models` indica a Django que busque `models.py` en el mismo directorio que `admin.py` . El código `admin.site.register()` le dice a Django que gestione nuestro modelo a través del sitio admin.

This code first imports the model we want to register, `Topic` . The dot in front of `models` tells Django to look for `models.py` in the same directory as `admin.py` . The code `admin.site.register()` tells Django to manage our model through the admin site.

Ahora utiliza la cuenta de superusuario para acceder al sitio admin. Ve a <http://localhost:8000/admin/> e introduce el nombre de usuario y la contraseña del superusuario que acabas de crear. Deberías ver una pantalla similar a la que se muestra en la [Figura 18-2](#) . Esta página te permite añadir nuevos usuarios y grupos, y modificar los existentes. También puedes trabajar con datos relacionados con el modelo `Topic` que acabamos de definir.

Now use the superuser account to access the admin site. Go to <http://localhost:8000/admin/> and enter the username and password for the superuser you just created. You should see a screen similar to the one shown in [Figure 18-2](#) . This page allows you to add new users and groups, and change existing ones. You can also work with data related to the `Topic` model that we just defined.



DESCRIPCIÓN DE LA IMAGEN (En español):

Administración de Django
Autenticación del sitio y autorización
Grupos
Agregar | Cambiar
Usuarios
Agregar | Cambiar
Registros de aprendizaje
Temas
Agregar | Cambiar
Acciones recientes
Mis acciones
Ninguna disponible
Bienvenido, LLM_ADMIN
Ver sitio / Cambiar contraseña / Cerrar sesión

Figura 18-2 : El sitio de administración con `Topic` incluido

Figure 18-2 : The admin site with `Topic` included

Nota (Note)

Si ves un mensaje en tu navegador indicando que la página web no está disponible, asegúrate de que aún tienes el servidor Django ejecutándose en una ventana de terminal. Si no es así, activa un entorno virtual y vuelve a emitir el comando `python manage.py runserver`. Si tienes problemas para ver tu proyecto en cualquier punto del proceso de desarrollo, cerrar cualquier terminal abierta y volver a emitir el comando `runserver` es un buen primer paso para solucionar el problema.

If you see a message in your browser that the web page is not available, make sure you still have the Django server running in a terminal window. If you don't, activate a virtual environment and reissue the command `python manage.py runserver`. If you're having trouble viewing your project at any point in the development process, closing any open terminals and reissuing the `runserver` command is a good first troubleshooting step.

Añadir temas (Adding Topics)

Ahora que `Topic` se ha registrado en el sitio de administración, vamos a añadir nuestro primer tema. Haz clic en **Temas** para ir a la página Temas, que está casi vacía, porque aún no tenemos temas que gestionar. Haz clic en **Añadir tema** y aparecerá un formulario para añadir un nuevo tema. Introduce `Chess` en la primera casilla y haz clic en **Guardar**. Volverás a la página de administración de Temas y verás el tema que acabas de crear.

Now that `Topic` has been registered with the admin site, let's add our first topic. Click **Topics** to go to the Topics page, which is mostly empty, because we have no topics to manage yet. Click **Add Topic**, and a form for adding a new topic appears. Enter `Chess` in the first box and click **Save**. You'll be sent back to the Topics admin page, and you'll see the topic you just created.

Vamos a crear un segundo tema para tener más datos con los que trabajar. Vuelve a hacer clic en **Añadir tema** e introduce `Rock Climbing`. Haz clic en **Guardar** y volverás a la página principal de Temas. Ahora verás que aparecen Ajedrez y Escalada.

Let's create a second topic so we'll have more data to work with. Click **Add Topic** again, and enter `Rock Climbing` . Click **Save** , and you'll be sent back to the main Topics page again. Now you'll see Chess and Rock Climbing listed.

Definir el modelo de entrada (*Defining the Entry Model*)

Para que un usuario pueda registrar lo que ha estado aprendiendo sobre ajedrez y escalada, necesitamos definir un modelo para los tipos de entradas que los usuarios pueden hacer en sus registros de aprendizaje. Cada entrada debe estar asociada a un tema concreto. Esta relación se denomina *many-to-one relationship* , lo que significa que muchas entradas pueden asociarse a un tema.

For a user to record what they've been learning about chess and rock climbing, we need to define a model for the kinds of entries users can make in their learning logs. Each entry needs to be associated with a particular topic. This relationship is called a *many-to-one relationship* , meaning many entries can be associated with one topic.

Aquí tienes el código del modelo `Entry` . Colócalo en tu archivo *models.py* :

Here's the code for the `Entry` model. Place it in your *models.py* file:

modelos.py

models.py

```
from django.db import models
```

```
class Topic(models.Model):  
    --snip--
```

```
❶ class Entry(models.Model):  
    """Something specific learned about a topic."""  
❷    topic = models.ForeignKey(Topic, on_delete=models.CASCADE)  
❸    text = models.TextField()  
    date_added = models.DateTimeField(auto_now_add=True)
```

```

❷ class Meta:
    verbose_name_plural = 'entries'

    def __str__(self):
        """Return a simple string representing the entry."""
❸     return f'{self.text[:50]}...'

```

La clase `Entry` hereda de la clase base `Model` de Django, al igual que `Topic` ❶. El primer atributo, `topic`, es una instancia de `ForeignKey` ❷. Un *foreign key* es un término de la base de datos; es una referencia a otro registro de la base de datos. Es el código que conecta cada entrada con un tema específico. A cada tema se le asigna un *key*, o ID, cuando se crea. Cuando Django necesita establecer una conexión entre dos datos, utiliza las claves asociadas a cada información. En breve utilizaremos estas conexiones para recuperar todas las entradas asociadas a un determinado tema. El argumento `on_delete=models.CASCADE` indica a Django que cuando se elimine un tema, se eliminen también todas las entradas asociadas a ese tema. Esto se conoce como *cascading delete*.

The `Entry` class inherits from Django's base `Model` class, just as `Topic` did ❶. The first attribute, `topic`, is a `ForeignKey` instance ❷. A *foreign key* is a database term; it's a reference to another record in the database. This is the code that connects each entry to a specific topic. Each topic is assigned a *key*, or ID, when it's created. When Django needs to establish a connection between two pieces of data, it uses the keys associated with each piece of information. We'll use these connections shortly to retrieve all the entries associated with a certain topic. The `on_delete=models.CASCADE` argument tells Django that when a topic is deleted, all the entries associated with that topic should be deleted as well. This is known as a *cascading delete*.

A continuación hay un atributo llamado `text`, que es una instancia de `TextField` ❸. Este tipo de campo no necesita un límite de tamaño, porque no queremos limitar el tamaño de las entradas individuales. El atributo `date_added` nos permite presentar las entradas en el orden en que fueron creadas, y colocar una marca de tiempo junto a cada entrada.

Next is an attribute called `text`, which is an instance of `TextField` ❸. This kind of field doesn't need a size limit, because we don't want to limit the size of individual entries. The `date_added` attribute allows us to present entries in the order they were created, and to place a timestamp next to each entry.

La clase `Meta` está anidada dentro de la clase `Entry` ④ . La clase `Meta` contiene información adicional para gestionar un modelo; aquí, nos permite establecer un atributo especial que indica a Django que utilice `Entries` cuando necesite hacer referencia a más de una entrada. Sin esto, Django se referiría a múltiples entradas como `Entrys` .

The `Meta` class is nested inside the `Entry` class ④ . The `Meta` class holds extra information for managing a model; here, it lets us set a special attribute telling Django to use `Entries` when it needs to refer to more than one entry. Without this, Django would refer to multiple entries as `Entrys` .

El método `__str__()` indica a Django qué información debe mostrar cuando hace referencia a entradas individuales. Como una entrada puede ser un cuerpo de texto largo, `__str__()` devuelve sólo los primeros 50 caracteres de `text` ⑤ . También añadimos una elipsis para aclarar que no siempre mostramos la entrada completa.

The `__str__()` method tells Django which information to show when it refers to individual entries. Because an entry can be a long body of text, `__str__()` returns just the first 50 characters of `text` ⑤ . We also add an ellipsis to clarify that we're not always displaying the entire entry.

Migrar el modelo de entrada (*Migrating the Entry Model*)

Como hemos añadido un nuevo modelo, tenemos que volver a migrar la base de datos. Este proceso te resultará bastante familiar: modificas `models.py` , ejecutas el comando `python manage.py makemigrations app_name` , y luego ejecutas el comando `python manage.py migrate` .

Because we've added a new model, we need to migrate the database again. This process will become quite familiar: you modify `models.py` , run the command `python manage.py makemigrations app_name` , and then run the command `python manage.py migrate` .

Migra la base de datos y comprueba el resultado introduciendo los siguientes comandos:

Migrate the database and check the output by entering the following commands:

```
(ll_env)learning_log$ python manage.py makemigrations learning_logs
```

```
Migrations for 'learning_logs':
```

```
❶ learning_logs/migrations/0002_entry.py
```

```
- Create model Entry
```

```
(ll_env)learning_log$ python manage.py migrate
```

```
Operations to perform:
```

```
--snip--
```

```
❷ Applying learning_logs.0002_entry... OK
```

Se genera una nueva migración llamada *0002_entry.py* , que indica a Django cómo modificar la base de datos para almacenar la información relacionada con el modelo `Entry` ❶ .

Cuando emitimos el comando `migrate` , vemos que Django aplicó esta migración y todo funcionó correctamente ❷ .

A new migration called *0002_entry.py* is generated, which tells Django how to modify the database to store information related to the model `Entry` ❶ . When we issue the `migrate` command, we see that Django applied this migration and everything worked properly ❷ .

Registro de entrada en el sitio de administración (*Registering Entry with the Admin Site*)

También necesitamos registrar el modelo `Entry` . Este es el aspecto que debería tener ahora *admin.py* :

We also need to register the `Entry` model. Here's what *admin.py* should look like now:

admin.py

admin.py

```
from django.contrib import admin
```

```
from .models import Topic, Entry
```

admin.site.register(Topic)
admin.site.register(Entry)

Vuelve a <http://localhost/admin/> , y deberías ver Entradas listadas bajo *Learning_Logs* . Haz clic en el enlace **Añadir** para Entradas, o haz clic en **Entradas** y luego elige **Añadir entrada** . Deberías ver una lista desplegable para seleccionar el tema para el que estás creando una entrada y un cuadro de texto para añadir una entrada. Selecciona **Ajedrez** en la lista desplegable y añade una entrada. Ésta es la primera entrada que hice:

Go back to <http://localhost/admin/> , and you should see Entries listed under *Learning_Logs* . Click the **Add** link for Entries, or click **Entries** and then choose **Add entry** . You should see a drop-down list to select the topic you're creating an entry for and a text box for adding an entry. Select **Chess** from the drop-down list, and add an entry. Here's the first entry I made:

La apertura es la primera parte de la partida, más o menos las diez primeras jugadas. En la apertura, conviene hacer tres cosas: sacar los alfiles y los caballos, intentar controlar el centro del tablero y enrocar el rey.

The opening is the first part of the game, roughly the first ten moves or so. In the opening, it's a good idea to do three things—bring out your bishops and knights, try to control the center of the board, and castle your king.

Por supuesto, esto son sólo directrices. Será importante aprender cuándo seguir estas directrices y cuándo hacer caso omiso de estas sugerencias.

Of course, these are just guidelines. It will be important to learn when to follow these guidelines and when to disregard these suggestions.

Cuando hagas clic en **Guardar** , volverás a la página principal de administración de las entradas. Aquí, verás la ventaja de utilizar `text[:50]` como representación de cadena para cada entrada; es mucho más fácil trabajar con varias entradas en la interfaz de administración si sólo ves la primera parte de una entrada, en lugar del texto completo de cada entrada.

When you click **Save** , you'll be brought back to the main admin page for entries. Here, you'll see the benefit of using `text[:50]` as the string representation for each entry; it's much easier to work with multiple entries in the admin interface if you see only the first part of an entry, rather than the entire text of each entry.

Haz una segunda entrada para Ajedrez y otra para Escalada, para que tengamos algunos datos iniciales. Aquí tienes una segunda entrada para Ajedrez:

Make a second entry for Chess and one entry for Rock Climbing so we have some initial data. Here's a second entry for Chess:

En la fase inicial de la partida, es importante sacar a relucir tus alfiles y caballos. Estas piezas son lo bastante poderosas y maniobrables como para desempeñar un papel importante en las jugadas iniciales de una partida.

In the opening phase of the game, it's important to bring out your bishops and knights. These pieces are powerful and maneuverable enough to play a significant role in the beginning moves of a game.

Y aquí tienes una primera entrada para Escalada:

And here's a first entry for Rock Climbing:

Uno de los conceptos más importantes en la escalada es mantener el peso sobre los pies tanto como sea posible. Existe el mito de que los escaladores pueden colgarse todo el día de los brazos. En realidad, los buenos escaladores han practicado formas específicas de mantener su peso sobre los pies siempre que sea posible.

One of the most important concepts in climbing is to keep your weight on your feet as much as possible. There's a myth that climbers can hang all day on their arms. In reality, good climbers have practiced specific ways of keeping their weight over their feet whenever possible.

Estas tres entradas nos darán algo con lo que trabajar mientras seguimos desarrollando el Registro de Aprendizaje.

These three entries will give us something to work with as we continue to develop Learning Log.

El caparazón de Django (*The Django Shell*)

Ahora que hemos introducido algunos datos, podemos examinarlos programáticamente a través de una sesión de terminal interactiva. Este entorno interactivo se llama *Django shell*, y es un entorno estupendo para probar y solucionar problemas de tu proyecto. Aquí tienes un ejemplo de sesión de terminal interactiva:

Now that we've entered some data, we can examine it programmatically through an interactive terminal session. This interactive environment is called the *Django shell*, and it's a great environment for testing and troubleshooting your project. Here's an example of an interactive shell session:

```
(ll_env)learning_log$ python manage.py shell
❶ >>> from learning_logs.models import Topic
>>> Topic.objects.all()
<QuerySet [<Topic: Chess>, <Topic: Rock Climbing>]>
```

El comando `python manage.py shell`, ejecutado en un entorno virtual activo, lanza un intérprete de Python que puedes utilizar para explorar los datos almacenados en la base de datos de tu proyecto. Aquí, importamos el modelo `Topic` del módulo `learning_logs.models` ❶. A continuación, utilizamos el método `Topic.objects.all()` para obtener todas las instancias del modelo `Topic`; la lista que se devuelve se denomina *queryset*.

The command `python manage.py shell`, run in an active virtual environment, launches a Python interpreter that you can use to explore the data stored in your project's database. Here, we import the model `Topic` from the `learning_logs.models` module ❶. We then use the method `Topic.objects.all()` to get all instances of the model `Topic`; the list that's returned is called a *queryset*.

Podemos hacer un bucle sobre un conjunto de consultas igual que lo haríamos sobre una lista. Así puedes ver el ID que se ha asignado a cada objeto tema:

We can loop over a queryset just as we'd loop over a list. Here's how you can see the ID that's been assigned to each topic object:

```
>>> topics = Topic.objects.all()
>>> for topic in topics:
...     print(topic.id, topic)
...
1 Chess
2 Rock Climbing
```

Asignamos el conjunto de consultas a `topics` y, a continuación, imprimimos el atributo `id` de cada tema y la representación de cadena de cada tema. Podemos ver que `Chess` tiene un ID de `1` y `Rock Climbing` tiene un ID de `2`.

We assign the queryset to `topics` and then print each topic's `id` attribute and the string representation of each topic. We can see that `Chess` has an ID of `1` and `Rock Climbing` has an ID of `2`.

Si conoces el ID de un objeto concreto, puedes utilizar el método `Topic.objects.get()` para recuperar ese objeto y examinar cualquier atributo que tenga. Veamos los valores `text` y `date_added` de `Chess`:

If you know the ID of a particular object, you can use the method `Topic.objects.get()` to retrieve that object and examine any attribute the object has. Let's look at the `text` and `date_added` values for `Chess`:

```
>>> t = Topic.objects.get(id=1)
>>> t.text
'Chess'
>>> t.date_added
datetime.datetime(2022, 5, 20, 3, 33, 36, 928759,
tzinfo=datetime.timezone.utc)
```


También podemos examinar las entradas relacionadas con un tema determinado. Anteriormente, definimos el atributo `topic` para el modelo `Entry`. Se trataba de un `ForeignKey`, una conexión entre cada entrada y un tema. Django puede utilizar esta conexión para obtener cada entrada relacionada con un tema determinado, de la siguiente manera:

We can also look at the entries related to a certain topic. Earlier, we defined the `topic` attribute for the `Entry` model. This was a `ForeignKey`, a connection between each entry and a topic. Django can use this connection to get every entry related to a certain topic, like this:

```
❶ >>> t.entry_set.all()
<QuerySet [<Entry: The opening is the first part of the game, roughly...>, <Entry:
In the opening phase of the game, it's important t...>]>
```

Para obtener datos a través de una relación de clave foránea, utiliza el nombre en minúsculas del modelo relacionado seguido de un guión bajo y la palabra `set` ❶. Por ejemplo, supongamos que tienes los modelos `Pizza` y `Topping`, y que `Topping` está relacionado con `Pizza` a través de una clave ajena. Si tu objeto se llama `my_pizza`, y representa una única pizza, puedes obtener todos los ingredientes de la pizza utilizando el código `my_pizza.topping_set.all()`.

To get data through a foreign key relationship, you use the lowercase name of the related model followed by an underscore and the word `set` ❶. For example, say you have the models `Pizza` and `Topping`, and `Topping` is related to `Pizza` through a foreign key. If your object is called `my_pizza`, representing a single pizza, you can get all of the pizza's toppings using the code `my_pizza.topping_set.all()`.

Utilizaremos esta sintaxis cuando empecemos a codificar las páginas que pueden solicitar los usuarios. El intérprete de comandos es realmente útil para asegurarte de que tu código recupera los datos que deseas. Si tu código funciona como esperas en el intérprete de comandos, también debería funcionar correctamente en los archivos de tu proyecto. Si tu código genera errores o no recupera los datos que esperas, es mucho más fácil solucionar los problemas de tu código en el sencillo entorno del shell que dentro de los archivos que generan las páginas web. No nos referiremos mucho al shell, pero deberías seguir utilizándolo para practicar el trabajo con la sintaxis de Django para acceder a los datos almacenados en el proyecto.

We'll use this syntax when we begin to code the pages users can request. The shell is really useful for making sure your code retrieves the data you want it to. If your code works as you expect it to in the shell, it should also work properly in the files within your project. If your code generates errors or doesn't retrieve the data you expect it to, it's much easier to troubleshoot your code in the simple shell environment than within the files that generate web pages. We won't refer to the shell much, but you should continue using it to practice working with Django's syntax for accessing the data stored in the project.

Cada vez que modifiques tus modelos, tendrás que reiniciar la shell para ver los efectos de esos cambios. Para salir de una sesión de shell, pulsa CTRL-D; en Windows, pulsa CTRL-Z y luego ENTER.

Each time you modify your models, you'll need to restart the shell to see the effects of those changes. To exit a shell session, press CTRL-D; on Windows, press CTRL-Z and then press ENTER.

Pruébalo tú mismo (*Try It Yourself*)

18-2. Entradas cortas: El método `__str__()` del modelo `Entry` actualmente añade una elipsis a cada instancia de `Entry` cuando Django lo muestra en el sitio de administración o en el shell. Añade una sentencia `if` al método `__str__()` que añada una elipsis sólo si la entrada tiene más de 50 caracteres. Utiliza el sitio de administración para añadir una entrada que tenga menos de 50 caracteres de longitud, y comprueba que no tiene una elipsis cuando se visualiza.

18-2. Short Entries: The `__str__()` method in the `Entry` model currently appends an ellipsis to every instance of `Entry` when Django shows it in the admin site or the shell. Add an `if` statement to the `__str__()` method that adds an ellipsis only if the entry is longer than 50 characters. Use the admin site to add an entry that's fewer than 50 characters in length, and check that it doesn't have an ellipsis when viewed.

18-3. La API de Django: Cuando escribes código para acceder a los datos de tu proyecto, estás escribiendo un *query*. Hojea la documentación para consultar tus datos en <https://docs.djangoproject.com/en/4.1/topics/db/queries>. Mucho de lo que veas te parecerá nuevo, pero te será bastante útil cuando empieces a trabajar en tus propios proyectos.

18-3. The Django API: When you write code to access the data in your project, you're writing a *query*. Skim through the documentation for querying your data at <https://docs.djangoproject.com/en/4.1/topics/db/queries>. Much of what you see will look new to you, but it will be quite useful as you start to work on your own projects.

18-4. Pizzería: Inicia un nuevo proyecto llamado `pizzeria_project` con una aplicación llamada `pizzas`. Define un modelo `Pizza` con un campo llamado `name`, que contendrá valores de nombre, como `Hawaiian` y `Meat Lovers`. Define un modelo llamado `Topping` con campos llamados `pizza` y `name`. El campo `pizza` debe ser una clave ajena a `Pizza`, y `name` debe poder contener valores como `pineapple`, `Canadian bacon` y `sausage`.

18-4. Pizzeria: Start a new project called `pizzeria_project` with an app called `pizzas`. Define a model `Pizza` with a field called `name`, which will hold name values, such as `Hawaiian` and `Meat Lovers`. Define a model called `Topping` with fields called `pizza` and `name`. The `pizza` field should be a foreign key to `Pizza`, and `name` should be able to hold values such as `pineapple`, `Canadian bacon`, and `sausage`.

Registra ambos modelos en el sitio de administración y utilízalo para introducir algunos nombres de pizza y sus ingredientes. Utiliza el intérprete de comandos para explorar los datos introducidos.

Register both models with the admin site, and use the site to enter some pizza names and toppings. Use the shell to explore the data you entered.

Crear páginas: La Página de Inicio del Registro de Aprendizaje (*Making Pages: The Learning Log Home Page*)

Hacer páginas web con Django consta de tres etapas: definir URLs, escribir vistas y escribir plantillas. Puedes hacerlas en cualquier orden, pero en este proyecto siempre empezaremos definiendo el patrón de URL. Un *URL pattern* describe la forma en que está dispuesta la URL. También le dice a Django qué debe buscar cuando hace coincidir una petición del navegador con la URL de un sitio, para saber qué página debe devolver.

Making web pages with Django consists of three stages: defining URLs, writing views, and writing templates. You can do these in any order, but in this project we'll always start by defining the URL pattern. A *URL pattern* describes the way the URL is laid out. It also tells Django what to look for when matching a browser request with a site URL, so it knows which page to return.

Cada URL se asigna a una vista concreta. La función *view* recupera y procesa los datos necesarios para esa página. La función de vista a menudo renderiza la página utilizando un *template*, que contiene la estructura general de la página. Para ver cómo funciona esto, vamos a crear la página de inicio del Registro de Aprendizaje. Definiremos la URL de la página de inicio, escribiremos su función de vista y crearemos una plantilla sencilla.

Each URL then maps to a particular view. The *view* function retrieves and processes the data needed for that page. The view function often renders the page using a *template*, which contains the overall structure of the page. To see how this works, let's make the home page for Learning Log. We'll define the URL for the home page, write its view function, and create a simple template.

Como sólo queremos asegurarnos de que Learning Log funciona como se supone que debe hacerlo, de momento crearemos una página sencilla. Es divertido dar estilo a una aplicación web que funciona cuando está completa; una aplicación que tiene buen aspecto pero no funciona bien no tiene sentido. Por ahora, la página de inicio sólo mostrará un título y una breve descripción.

Because we just want to ensure that Learning Log works as it's supposed to, we'll make a simple page for now. A functioning web app is fun to style when it's complete; an app that looks good but doesn't work well is pointless. For now, the home page will display only a title and a brief description.

Asignar una URL (*Mapping a URL*)

Los usuarios solicitan las páginas introduciendo URL en un navegador y haciendo clic en los enlaces, así que tendremos que decidir qué URL son necesarias. La URL de la página de inicio es lo primero: es la URL base que la gente utiliza para acceder al proyecto. Por el momento, la URL base, `http://localhost:8000/`, devuelve el sitio Django predeterminado que nos permite saber que el proyecto se ha configurado correctamente. Cambiaremos esto

asignando la URL base a la página de inicio de Learning Log.

Users request pages by entering URLs into a browser and clicking links, so we'll need to decide what URLs are needed. The home page URL is first: it's the base URL people use to access the project. At the moment, the base URL, `http://localhost:8000/`, returns the default Django site that lets us know the project was set up correctly. We'll change this by mapping the base URL to Learning Log's home page.

En la carpeta principal `ll_project`, abre el archivo `urls.py`. Deberías ver el siguiente código:

In the main `ll_project` folder, open the file `urls.py`. You should see the following code:

ll_project/urls.py

ll_project/urls.py

```
❶ from django.contrib import admin
from django.urls import path

❷ urlpatterns = [
❸     path('admin/', admin.site.urls),
]
```

Las dos primeras líneas importan el módulo `admin` y una función para construir rutas URL ❶. El cuerpo del archivo define la variable `urlpatterns` ❷. En este archivo `urls.py`, que define las URL para el proyecto en su conjunto, la variable `urlpatterns` incluye conjuntos de URL de las aplicaciones del proyecto. La lista incluye el módulo `admin.site.urls`, que define todas las URL que se pueden solicitar desde el sitio de administración ❸.

The first two lines import the `admin` module and a function to build URL paths ❶. The body of the file defines the `urlpatterns` variable ❷. In this `urls.py` file, which defines URLs for the project as a whole, the `urlpatterns` variable includes sets of URLs from the apps in

the project. The list includes the module `admin.site.urls` , which defines all the URLs that can be requested from the admin site ❸ .

Necesitamos incluir las URL de `learning_logs` , así que añade lo siguiente:

We need to include the URLs for `learning_logs` , so add the following:

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path("", include('learning_logs.urls')),
]
```

Hemos importado la función `include()` , y también hemos añadido una línea para incluir el módulo `learning_logs.urls` .

We've imported the `include()` function, and we've also added a line to include the module `learning_logs.urls` .

Por defecto, `urls.py` está en la carpeta `ll_project` ; ahora necesitamos crear un segundo archivo `urls.py` en la carpeta `learning_logs` . Crea un nuevo archivo Python, guárdalo como `urls.py` en `learning_logs` , e introduce en él este código:

The default `urls.py` is in the `ll_project` folder; now we need to make a second `urls.py` file in the `learning_logs` folder. Create a new Python file, save it as `urls.py` in `learning_logs` , and enter this code into it:

learning_logs/urls.py

learning_logs/urls.py

❶ `"""Defines URL patterns for learning_logs."""`

❷ `from django.urls import path`

❸ `from . import views`

❹ `app_name = 'learning_logs'`

❺ `urlpatterns = [`

`# Home page`

❻ `path("", views.index, name='index'),`

`]`

Para dejar claro en qué *urls.py* estamos trabajando, añadimos un docstring al principio del archivo ❶ . A continuación, importamos la función `path` , necesaria para asignar URL a vistas ❷ . También importamos el módulo `views` ❸ ; el punto indica a Python que importe el módulo *views.py* del mismo directorio que el módulo actual *urls.py* . La variable `app_name` ayuda a Django a distinguir este archivo *urls.py* de los archivos del mismo nombre de otras aplicaciones del proyecto ❹ . La variable `urlpatterns` de este módulo es una lista de páginas individuales que pueden solicitarse a la aplicación *learning_logs* ❺ .

To make it clear which *urls.py* we're working in, we add a docstring at the beginning of the file ❶ . We then import the `path` function, which is needed when mapping URLs to views ❷ . We also import the `views` module ❸ ; the dot tells Python to import the *views.py* module from the same directory as the current *urls.py* module. The variable `app_name` helps Django distinguish this *urls.py* file from files of the same name in other apps within the project ❹ . The variable `urlpatterns` in this module is a list of individual pages that can be requested from the *learning_logs* app ❺ .

El patrón URL real es una llamada a la función `path()` , que toma tres argumentos ❻ . El primer argumento es una cadena que ayuda a Django a enrutar correctamente la solicitud actual. Django recibe la URL solicitada e intenta dirigir la solicitud a una vista. Para ello, busca en todos los patrones de URL que hemos definido uno que coincida con la solicitud actual. Django ignora la URL base del proyecto (`http://localhost:8000/`), por lo que la cadena vacía (`"`) coincide con la URL base. Cualquier otra URL no coincidirá con este patrón, y Django devolverá una página de error si la URL solicitada no coincide con ningún patrón de URL existente.

The actual URL pattern is a call to the `path()` function, which takes three arguments ⑥ . The first argument is a string that helps Django route the current request properly. Django receives the requested URL and tries to route the request to a view. It does this by searching all the URL patterns we've defined to find one that matches the current request. Django ignores the base URL for the project (`http://localhost:8000/`), so the empty string (`"`) matches the base URL. Any other URL won't match this pattern, and Django will return an error page if the URL requested doesn't match any existing URL patterns.

El segundo argumento de `path()` ⑥ especifica a qué función llamar en `views.py` . Cuando una URL solicitada coincide con el patrón que estamos definiendo, Django llama a la función `index()` de `views.py` . (Escribiremos esta función de vista en la siguiente sección.) El tercer argumento proporciona el nombre *index* a este patrón de URL para que podamos referirnos a él más fácilmente en otros archivos del proyecto. Siempre que queramos proporcionar un enlace a la página de inicio, utilizaremos este nombre en lugar de escribir una URL.

The second argument in `path()` ⑥ specifies which function to call in `views.py` . When a requested URL matches the pattern we're defining, Django calls the `index()` function from `views.py` . (We'll write this view function in the next section.) The third argument provides the name *index* for this URL pattern so we can refer to it more easily in other files throughout the project. Whenever we want to provide a link to the home page, we'll use this name instead of writing out a URL.

Escribir una vista (*Writing a View*)

Una función de vista recibe información de una solicitud, prepara los datos necesarios para generar una página y, a continuación, devuelve los datos al navegador. A menudo lo hace utilizando una plantilla que define el aspecto que tendrá la página.

A view function takes in information from a request, prepares the data needed to generate a page, and then sends the data back to the browser. It often does this by using a template that defines what the page will look like.

El archivo `views.py` en `learning_logs` se generó automáticamente cuando ejecutamos el comando `python manage.py startapp` . Esto es lo que hay ahora en `views.py` :

The file *views.py* in *learning_logs* was generated automatically when we ran the command `python manage.py startapp` . Here's what's in *views.py* right now:

views.py

views.py

```
from django.shortcuts import render
```

```
# Create your views here.
```

Actualmente, este archivo sólo importa la función `render()` , que renderiza la respuesta basándose en los datos proporcionados por views. Abre *views.py* y añade el siguiente código para la página de inicio:

Currently, this file just imports the `render()` function, which renders the response based on the data provided by views. Open *views.py* and add the following code for the home page:

```
from django.shortcuts import render
```

```
def index(request):
```

```
    """The home page for Learning Log."""
```

```
    return render(request, 'learning_logs/index.xhtmll')
```

Cuando una petición de URL coincide con el patrón que acabamos de definir, Django busca una función llamada `index()` en el archivo *views.py* . A continuación, Django pasa el objeto `request` a esta función de vista. En este caso, no necesitamos procesar ningún dato para la página, por lo que el único código de la función es una llamada a `render()` . La función `render()` pasa aquí dos argumentos: el objeto original `request` y una plantilla que puede utilizar para construir la página. Escribamos esta plantilla.

When a URL request matches the pattern we just defined, Django looks for a function called `index()` in the *views.py* file. Django then passes the `request` object to this view function. In this case, we don't need to process any data for the page, so the only code in the function is

a call to `render()` . The `render()` function here passes two arguments: the original request object and a template it can use to build the page. Let's write this template.

Escribir una plantilla (*Writing a Template*)

La plantilla define el aspecto que debe tener la página, y Django rellena los datos relevantes cada vez que se solicita la página. Una plantilla te permite acceder a cualquier dato proporcionado por la vista. Como nuestra vista para la página de inicio no proporciona datos, esta plantilla es bastante sencilla.

The template defines what the page should look like, and Django fills in the relevant data each time the page is requested. A template allows you to access any data provided by the view. Because our view for the home page provides no data, this template is fairly simple.

Dentro de la carpeta *learning_logs* , crea una nueva carpeta llamada *templates* . Dentro de la carpeta *templates* , crea otra carpeta llamada *learning_logs* . Esto puede parecer un poco redundante (tenemos una carpeta llamada *learning_logs* dentro de una carpeta llamada *templates* dentro de una carpeta llamada *learning_logs*), pero establece una estructura que Django puede interpretar sin ambigüedades, incluso en el contexto de un gran proyecto que contenga muchas aplicaciones individuales. Dentro de la carpeta *learning_logs* , crea un nuevo archivo llamado *index.html* . La ruta del archivo será *ll_project/learning_logs/templates/learning_logs/index.html* . Introduce el siguiente código en ese archivo:

Inside the *learning_logs* folder, make a new folder called *templates* . Inside the *templates* folder, make another folder called *learning_logs* . This might seem a little redundant (we have a folder named *learning_logs* inside a folder named *templates* inside a folder named *learning_logs*), but it sets up a structure that Django can interpret unambiguously, even in the context of a large project containing many individual apps. Inside the inner *learning_logs* folder, make a new file called *index.html* . The path to the file will be *ll_project/learning_logs/templates/learning_logs/index.html* . Enter the following code into that file:

index.html

index.html

<p>Learning Log</p>

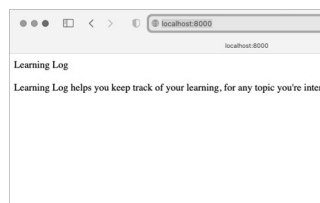
<p>Learning Log helps you keep track of your learning, for any topic you're interested in.</p>

Se trata de un archivo muy sencillo. Si no estás familiarizado con HTML, las etiquetas <p></p> significan párrafos. La etiqueta <p> abre un párrafo, y la etiqueta </p> lo cierra. Tenemos dos párrafos: el primero actúa como título, y el segundo describe lo que los usuarios pueden hacer con Learning Log.

This is a very simple file. If you're not familiar with HTML, the <p></p> tags signify paragraphs. The <p> tag opens a paragraph, and the </p> tag closes a paragraph. We have two paragraphs: the first acts as a title, and the second describes what users can do with Learning Log.

Ahora, cuando solicites la URL base del proyecto, *http://localhost:8000/*, deberías ver la página que acabamos de construir en lugar de la página por defecto de Django. Django tomará la URL solicitada, y esa URL coincidirá con el patrón " "; entonces Django llamará a la función `views.index()`, que renderizará la página utilizando la plantilla contenida en *index.xhtml*. La [Figura 18-3](#) muestra la página resultante.

Now when you request the project's base URL, *http://localhost:8000/*, you should see the page we just built instead of the default Django page. Django will take the requested URL, and that URL will match the pattern " "; then Django will call the function `views.index()`, which will render the page using the template contained in *index.xhtml*. [Figure 18-3](#) shows the resulting page.



DESCRIPCIÓN DE LA
IMAGEN (En español):

Registro de
Aprendizaje El Registro
de Aprendizaje te
ayuda a llevar un
control de tu
aprendizaje, para
cualquier tema en el
que estés interesado.

Figura 18-3 : Página de inicio del Registro de Aprendizaje

Figure 18-3 : The home page for Learning Log

Aunque pueda parecer un proceso complicado para crear una página, esta separación entre URL, vistas y plantillas funciona bastante bien. Te permite pensar en cada aspecto de un proyecto por separado. En proyectos más grandes, permite a las personas que trabajan en el proyecto centrarse en las áreas en las que son más fuertes. Por ejemplo, un especialista en bases de datos puede centrarse en los modelos, un programador en el código de las vistas y un especialista en frontend en las plantillas.

Although it might seem like a complicated process for creating one page, this separation between URLs, views, and templates works quite well. It allows you to think about each aspect of a project separately. In larger projects, it allows individuals working on the project to focus on the areas in which they're strongest. For example, a database specialist can focus on the models, a programmer can focus on the view code, and a frontend specialist can focus on the templates.

Nota (Note)

Es posible que veas el siguiente mensaje de error:

You might see the following error message:

ModuleNotFoundError: No module named 'learning_logs.urls'

Si es así, detén el servidor de desarrollo pulsando CTRL-C en la ventana del terminal donde emitiste el comando `runserver` . A continuación, vuelve a emitir el comando `python manage.py runserver` . Deberías poder ver la página de inicio. Cada vez que te encuentres con un error como éste, intenta detener y reiniciar el servidor.

If you do, stop the development server by pressing CTRL-C in the terminal window where you issued the `runserver` command. Then reissue the command `python manage.py runserver` . You should be able to see the home page. Anytime you run into an error like this, try stopping and restarting the server.

Pruébalo tú mismo (*Try It Yourself*)

18-5. Planificador de comidas: Piensa en una aplicación que ayude a la gente a planificar sus comidas a lo largo de la semana. Crea una nueva carpeta llamada *meal_planner* , e inicia un nuevo proyecto Django dentro de esta carpeta. A continuación, crea una nueva aplicación llamada *meal_plans* . Haz una página de inicio sencilla para este proyecto.

18-5. Meal Planner: Consider an app that helps people plan their meals throughout the week. Make a new folder called *meal_planner* , and start a new Django project inside this folder. Then make a new app called *meal_plans* . Make a simple home page for this project.

18-6. Página de inicio de la Pizzería: Añade una página de inicio al proyecto Pizzería que comenzaste en el Ejercicio 18-4 (página 388) .

18-6. Pizzeria Home Page: Add a home page to the Pizzeria project you started in Exercise 18-4 (page 388) .

Construir Páginas Adicionales (*Building Additional Pages*)

Ahora que hemos establecido una rutina para construir una página, podemos empezar a construir el proyecto Registro de Aprendizaje. Construiremos dos páginas que muestren datos: una página que enumere todos los temas y una página que muestre todas las entradas de un tema concreto. Para cada página, especificaremos un patrón URL, escribiremos una función de vista y escribiremos una plantilla. Pero antes, crearemos una plantilla base de la que podrán heredar todas las plantillas del proyecto.

Now that we've established a routine for building a page, we can start to build out the Learning Log project. We'll build two pages that display data: a page that lists all topics and a page that shows all the entries for a particular topic. For each page, we'll specify a URL pattern, write a view function, and write a template. But before we do this, we'll create a base template that all templates in the project can inherit from.

Herencia de plantillas (*Template Inheritance*)

Al construir un sitio web, algunos elementos tendrán que repetirse en cada página. En lugar de escribir estos elementos directamente en cada página, puedes escribir una plantilla base que contenga los elementos repetidos y luego hacer que cada página herede de la base. Este enfoque te permite centrarte en desarrollar los aspectos únicos de cada página, y hace que sea mucho más fácil cambiar el aspecto general del proyecto.

When building a website, some elements will need to be repeated on each page. Rather than writing these elements directly into each page, you can write a base template containing the repeated elements and then have each page inherit from the base. This approach lets you focus on developing the unique aspects of each page, and makes it much easier to change the overall look and feel of the project.

La plantilla padre (*The Parent Template*)

Crearemos una plantilla llamada *base.xhtml* en el mismo directorio que *index.xhtml*. Este archivo contendrá elementos comunes a todas las páginas; todas las demás plantillas heredarán de *base.xhtml*. El único elemento que queremos repetir en cada página en este momento es el título de la parte superior. Como incluiremos esta plantilla en todas las páginas, hagamos que el título sea un enlace a la página de inicio:

We'll create a template called *base.xhtml* in the same directory as *index.xhtml*. This file will contain elements common to all pages; every other template will inherit from *base.xhtml*. The only element we want to repeat on each page right now is the title at the top. Because we'll include this template on every page, let's make the title a link to the home page:

base.xhtml

base.xhtml

```
<p>  
❶ <a href="{% url 'learning_logs:index' %}">Learning Log</a>  
</p>
```

② {% block content %}{% endblock content %}

La primera parte de este archivo crea un párrafo que contiene el nombre del proyecto, que también actúa como enlace a la página de inicio. Para generar un enlace, utilizamos una etiqueta *template tag*, que se indica mediante llaves y signos de porcentaje ({% %}). Una etiqueta de plantilla genera información que se mostrará en una página. La etiqueta de plantilla {% url 'learning_logs:index' %} que se muestra aquí genera una URL que coincide con el patrón de URL definido en *learning_logs/urls.py* con el nombre 'index' ① . En este ejemplo, *learning_logs* es el *namespace* y *index* es un patrón de URL de nombre único en ese espacio de nombres. El espacio de nombres procede del valor que asignamos a *app_name* en el archivo *learning_logs/urls.py*.

The first part of this file creates a paragraph containing the name of the project, which also acts as a home page link. To generate a link, we use a *template tag*, which is indicated by braces and percent signs ({% %}). A template tag generates information to be displayed on a page. The template tag {% url 'learning_logs:index' %} shown here generates a URL matching the URL pattern defined in *learning_logs/urls.py* with the name 'index' ① . In this example, *learning_logs* is the *namespace* and *index* is a uniquely named URL pattern in that namespace. The namespace comes from the value we assigned to *app_name* in the *learning_logs/urls.py* file.

En una página HTML simple, un enlace está rodeado por el *anchor tag* <a> :

In a simple HTML page, a link is surrounded by the *anchor tag* <a> :

```
<a href="link_url">link text</a>
```

Hacer que la etiqueta de plantilla genere la URL por nosotros hace que sea mucho más fácil mantener nuestros enlaces actualizados. Sólo tenemos que cambiar el patrón de URL en *urls.py*, y Django insertará automáticamente la URL actualizada la próxima vez que se solicite la página. Todas las páginas de nuestro proyecto heredarán de *base.xhtml*, por lo que, a partir de ahora, todas las páginas tendrán un enlace a la página de inicio.

Having the template tag generate the URL for us makes it much easier to keep our links up to date. We only need to change the URL pattern in *urls.py*, and Django will automatically insert the updated URL the next time the page is requested. Every page in our project will inherit from *base.xhtml*, so from now on, every page will have a link back to the home page.

En la última línea, insertamos un par de etiquetas `block` ② . Este bloque, llamado `content` , es un marcador de posición; la plantilla hija definirá el tipo de información que va en el bloque `content` .

On the last line, we insert a pair of `block` tags ② . This block, named `content` , is a placeholder; the child template will define the kind of information that goes in the `content` block.

Una plantilla hija no tiene por qué definir todos los bloques de su padre, así que puedes reservar espacio en las plantillas padre para tantos bloques como quieras; la plantilla hija sólo utilizará los que necesite.

A child template doesn't have to define every block from its parent, so you can reserve space in parent templates for as many blocks as you like; the child template uses only as many as it needs.

Nota (Note)

En el código Python, casi siempre utilizamos cuatro espacios cuando hacemos sangrías. Los archivos de plantilla suelen tener más niveles de anidamiento que los archivos Python, por lo que es habitual utilizar sólo dos espacios para cada nivel de sangría.

In Python code, we almost always use four spaces when we indent. Template files tend to have more levels of nesting than Python files, so it's common to use only two spaces for each indentation level.

La plantilla hija (The Child Template)

Ahora tenemos que reescribir `index.xhtml` para que herede de `base.xhtml` . Añade el siguiente código a `index.xhtml` :

Now we need to rewrite *index.xhtml* to inherit from *base.xhtml* . Add the following code to *index.xhtml* :

index.xhtml

index.xhtml

❶ {% extends 'learning_logs/base.xhtml' %}

❷ {% block content %}

<p>Learning Log helps you keep track of your learning, for any topic you're interested in.</p>

❸ {% endblock content %}

Si comparas esto con el original *index.xhtml* , puedes ver que hemos sustituido el título Learning Log por el código para heredar de una plantilla padre ❶ . Una plantilla hija debe tener una etiqueta `{% extends %}` en la primera línea para indicar a Django de qué plantilla padre debe heredar. El archivo *base.xhtml* forma parte de *learning_logs* , por lo que incluimos *learning_logs* en la ruta a la plantilla padre. Esta línea recoge todo lo que contiene la plantilla *base.xhtml* y permite a *index.xhtml* definir lo que va en el espacio reservado por el bloque `content` .

If you compare this to the original *index.xhtml* , you can see that we've replaced the Learning Log title with the code for inheriting from a parent template ❶ . A child template must have an `{% extends %}` tag on the first line to tell Django which parent template to inherit from. The file *base.xhtml* is part of *learning_logs* , so we include *learning_logs* in the path to the parent template. This line pulls in everything contained in the *base.xhtml* template and allows *index.xhtml* to define what goes in the space reserved by the `content` block.

Definimos el bloque de contenido insertando una etiqueta `{% block %}` con el nombre `content` ❷ . Todo lo que no estamos heredando de la plantilla padre va dentro del bloque `content` . Aquí, ese es el párrafo que describe el proyecto Learning Log. Indicamos que hemos terminado de definir el contenido utilizando una etiqueta `{% endblock content %}` ❸ . La etiqueta `{% endblock %}` no requiere un nombre, pero si una plantilla crece hasta contener varios bloques, puede ser útil saber exactamente qué bloque está terminando.

We define the content block by inserting a `{% block %}` tag with the name `content` ② . Everything that we aren't inheriting from the parent template goes inside the `content` block. Here, that's the paragraph describing the Learning Log project. We indicate that we're finished defining the content by using an `{% endblock content %}` tag ③ . The `{% endblock %}` tag doesn't require a name, but if a template grows to contain multiple blocks, it can be helpful to know exactly which block is ending.

Puedes empezar a ver el beneficio de la herencia de plantillas: en una plantilla hija, sólo necesitamos incluir contenido que sea exclusivo de esa página. Esto no sólo simplifica cada plantilla, sino que también facilita mucho la modificación de la página. Para modificar un elemento común a muchas páginas, sólo tienes que modificar la plantilla padre. Tus cambios se trasladarán a todas las páginas que hereden de esa plantilla. En un proyecto que incluya decenas o cientos de páginas, esta estructura puede hacer mucho más fácil y rápido mejorar tu sitio.

You can start to see the benefit of template inheritance: in a child template, we only need to include content that's unique to that page. This not only simplifies each template, but also makes it much easier to modify the site. To modify an element common to many pages, you only need to modify the parent template. Your changes are then carried over to every page that inherits from that template. In a project that includes tens or hundreds of pages, this structure can make it much easier and faster to improve your site.

En un proyecto grande, es habitual tener una plantilla padre llamada *base.xhtml* para todo el sitio y plantillas padre para cada sección principal del sitio. Todas las plantillas de sección heredan de *base.xhtml* , y cada página del sitio hereda de una plantilla de sección. De esta forma puedes modificar fácilmente el aspecto del sitio en su conjunto, de cualquier sección del sitio o de cualquier página individual. Esta configuración proporciona una forma muy eficaz de trabajar, y te anima a actualizar tu proyecto de forma constante a lo largo del tiempo.

In a large project, it's common to have one parent template called *base.xhtml* for the entire site and parent templates for each major section of the site. All the section templates inherit from *base.xhtml* , and each page in the site inherits from a section template. This way you can easily modify the look and feel of the site as a whole, any section in the site, or any individual page. This configuration provides a very efficient way to work, and encourages you to steadily update your project over time.

La Página de Temas (*The Topics Page*)

Ahora que tenemos un enfoque eficiente para construir páginas, podemos centrarnos en nuestras dos siguientes páginas: la página de temas generales y la página para mostrar las entradas de un único tema. La página de temas mostrará todos los temas que han creado los usuarios, y es la primera página que implicará trabajar con datos.

Now that we have an efficient approach to building pages, we can focus on our next two pages: the general topics page and the page to display entries for a single topic. The topics page will show all topics that users have created, and it's the first page that will involve working with data.

El patrón URL de los temas (*The Topics URL Pattern*)

En primer lugar, definimos la URL de la página de temas. Es habitual elegir un fragmento de URL simple que refleje el tipo de información presentada en la página. Utilizaremos la palabra *topics*, por lo que la URL `http://localhost:8000/topics/` devolverá esta página. Así es como modificamos `learning_logs/urls.py`:

First, we define the URL for the topics page. It's common to choose a simple URL fragment that reflects the kind of information presented on the page. We'll use the word *topics*, so the URL `http://localhost:8000/topics/` will return this page. Here's how we modify `learning_logs/urls.py`:

learning_logs/urls.py

learning_logs/urls.py

```
"""Defines URL patterns for learning_logs."""
--snip--
urlpatterns = [
    # Home page
    path("", views.index, name='index'),
    # Page that shows all topics.
```

```
    path('topics/', views.topics, name='topics'),  
]
```

El nuevo patrón de URL es la palabra *topics* , seguida de una barra oblicua. Cuando Django examine una URL solicitada, este patrón coincidirá con cualquier URL que tenga la URL base seguida de *topics* . Puedes incluir u omitir una barra diagonal al final, pero no puede haber nada más después de la palabra *topics* , o el patrón no coincidirá. Cualquier solicitud con una URL que coincida con este patrón se pasará a la función `topics()` en *views.py* .

The new URL pattern is the word *topics* , followed by a forward slash. When Django examines a requested URL, this pattern will match any URL that has the base URL followed by *topics* . You can include or omit a forward slash at the end, but there can't be anything else after the word *topics* , or the pattern won't match. Any request with a URL that matches this pattern will then be passed to the function `topics()` in *views.py* .

La vista Temas (The Topics View)

La función `topics()` necesita recuperar algunos datos de la base de datos y enviarlos a la plantilla. Añade lo siguiente a *views.py* :

The `topics()` function needs to retrieve some data from the database and send it to the template. Add the following to *views.py* :

vistas.py

views.py

```
from django.shortcuts import render
```

```
❶ from .models import Topic
```

```
def index(request):
```

```
    --snip--
```

```
❷ def topics(request):  
    """Show all topics."""
```

```
③ topics = Topic.objects.order_by('date_added')
④ context = {'topics': topics}
⑤ return render(request, 'learning_logs/topics.xhtml', context)
```

Primero importamos el modelo asociado a los datos que necesitamos ❶ . La función `topics()` necesita un parámetro: el objeto `request` que Django ha recibido del servidor ❷ . Consultamos la base de datos pidiendo los objetos `Topic` ordenados por el atributo `date_added` ❸ . Asignamos el conjunto de consultas resultante a `topics` .

We first import the model associated with the data we need ❶ . The `topics()` function needs one parameter: the `request` object Django received from the server ❷ . We query the database by asking for the `Topic` objects, sorted by the `date_added` attribute ❸ . We assign the resulting queryset to `topics` .

A continuación, definimos un contexto que enviaremos a la plantilla ❹ . Un *context* es un diccionario en el que las claves son nombres que utilizaremos en la plantilla para acceder a los datos que queremos, y los valores son los datos que necesitamos enviar a la plantilla. En este caso, hay un par clave-valor, que contiene el conjunto de temas que mostraremos en la página. Cuando construimos una página que utiliza datos, llamamos a `render()` con el objeto `request` , la plantilla que queremos utilizar, y el diccionario `context` ❺ .

We then define a context that we'll send to the template ❹ . A *context* is a dictionary in which the keys are names we'll use in the template to access the data we want, and the values are the data we need to send to the template. In this case, there's one key-value pair, which contains the set of topics we'll display on the page. When building a page that uses data, we call `render()` with the `request` object, the template we want to use, and the context dictionary ❺ .

La plantilla de temas (The Topics Template)

La plantilla para la página de temas recibe el diccionario `context` , para que la plantilla pueda utilizar los datos que proporciona `topics()` . Crea un archivo llamado *topics.xhtml* en el mismo directorio que *index.xhtml* . Así es como podemos mostrar los temas en la plantilla:

The template for the topics page receives the `context` dictionary, so the template can use the data that `topics()` provides. Make a file called *topics.xhtml* in the same directory as *index.xhtml* . Here's how we can display the topics in the template:

temas.xhtml

topics.xhtml

```
{% extends 'learning_logs/base.xhtml' %}
```

```
{% block content %}
```

```
<p>Topics</p>
```

```
❶ <ul>
❷   {% for topic in topics %}
❸     <li>{{ topic.text }}</li>
❹   {% empty %}
     <li>No topics have been added yet.</li>
❺   {% endfor %}
❻ </ul>
```

```
{% endblock content %}
```

Utilizamos la etiqueta `{% extends %}` para heredar de *base.xhtml* , igual que hicimos en la página de inicio, y luego abrimos un bloque `content` . El cuerpo de esta página contiene una lista con viñetas de los temas que se han introducido. En HTML estándar, una lista con viñetas se denomina *unordered list* y se indica mediante las etiquetas `<ul></ul>` . La etiqueta de apertura `<ul>` inicia la lista con viñetas de temas ❶ .

We use the `{% extends %}` tag to inherit from *base.xhtml* , just as we did on the home page, and then we open a `content` block. The body of this page contains a bulleted list of the topics that have been entered. In standard HTML, a bulleted list is called an *unordered list* and is indicated by the tags `<ul></ul>` . The opening tag `<ul>` begins the bulleted list of topics ❶ .

A continuación utilizamos una etiqueta de plantilla que equivale a un bucle `for`, que recorre la lista `topics` desde el diccionario `context` ❷. El código utilizado en las plantillas difiere del de Python en algunos aspectos importantes. Python utiliza la sangría para indicar qué líneas de una sentencia `for` forman parte de un bucle. En una plantilla, cada bucle `for` necesita una etiqueta `{% endfor %}` explícita que indique dónde se produce el final del bucle. Por tanto, en una plantilla, verás bucles escritos así:

Next we use a template tag that's equivalent to a `for` loop, which loops through the list `topics` from the `context` dictionary ❷. The code used in templates differs from Python in some important ways. Python uses indentation to indicate which lines of a `for` statement are part of a loop. In a template, every `for` loop needs an explicit `{% endfor %}` tag indicating where the end of the loop occurs. So in a template, you'll see loops written like this:

```
{% for item in list %}
  do something with each item
{% endfor %}
```

Dentro del bucle, queremos convertir cada tema en un elemento de la lista con viñetas. Para imprimir una variable en una plantilla, encierra el nombre de la variable entre llaves dobles. Las llaves no aparecerán en la página; sólo indican a Django que estamos utilizando una variable de plantilla. Así, el código `{{ topic.text }}` ❸ será sustituido por el valor del atributo `text` del tema actual en cada pasada por el bucle. La etiqueta HTML `<li></li>` indica un *list item*. Cualquier cosa entre estas etiquetas, dentro de un par de etiquetas `<ul></ul>`, aparecerá como un elemento con viñetas en la lista.

Inside the loop, we want to turn each topic into an item in the bulleted list. To print a variable in a template, wrap the variable name in double braces. The braces won't appear on the page; they just indicate to Django that we're using a template variable. So the code `{{ topic.text }}` ❸ will be replaced by the value of the current topic's `text` attribute on each pass through the loop. The HTML tag `<li></li>` indicates a *list item*. Anything between these tags, inside a pair of `<ul></ul>` tags, will appear as a bulleted item in the list.

También utilizamos la etiqueta de plantilla `{% empty %}` ❹, que indica a Django qué hacer si no hay elementos en la lista. En este caso, imprimimos un mensaje informando al usuario de que aún no se ha añadido ningún tema. Las dos últimas líneas cierran el bucle `for` ❺ y luego cierran la lista con viñetas ❻.

We also use the `{% empty %}` template tag ❹, which tells Django what to do if there are no items in the list. In this case, we print a message informing the user that no topics have been added yet. The last two lines close out the `for` loop ❺ and then close out the bulleted list ❻.

Ahora tenemos que modificar la plantilla base para incluir un enlace a la página de temas. Añade el siguiente código a *base.xhtml* :

Now we need to modify the base template to include a link to the topics page. Add the following code to *base.xhtml* :

base.xhtml

base.xhtml

```
<p>
❶ <a href="{% url 'learning_logs:index' %}">Learning Log</a> -
❷ <a href="{% url 'learning_logs:topics' %}">Topics</a>
</p>
```

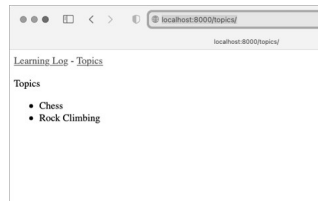
```
{% block content %}{% endblock content %}
```

Añadimos un guión después del enlace a la página de inicio ❶, y a continuación volvemos a añadir un enlace a la página de temas utilizando la etiqueta de plantilla `{% url %}` ❷. Esta línea indica a Django que genere un enlace que coincida con el patrón de URL con el nombre 'topics' en *learning_logs/urls.py*.

We add a dash after the link to the home page ❶, and then add a link to the topics page using the `{% url %}` template tag again ❷. This line tells Django to generate a link matching the URL pattern with the name 'topics' in *learning_logs/urls.py*.

Ahora, cuando actualices la página de inicio en tu navegador, verás un enlace a Temas. Cuando hagas clic en el enlace, verás una página similar a la de [la Figura 18-4](#).

Now when you refresh the home page in your browser, you'll see a Topics link. When you click the link, you'll see a page that looks similar to [Figure 18-4](#).



DESCRIPCIÓN DE LA
IMAGEN (En español):

Registro de
Aprendizaje - Temas
Temas - Ajedrez -
Escalada en Roca

Figura 18-4 : Página de temas

Figure 18-4 : The topics page

[Páginas de temas individuales \(Individual Topic Pages\)](#)

A continuación, necesitamos crear una página que pueda centrarse en un único tema, mostrando el nombre del tema y todas las entradas de ese tema. Definiremos un nuevo patrón URL, escribiremos una vista y crearemos una plantilla. También modificaremos la página de temas para que cada elemento de la lista con viñetas enlace a su correspondiente página de temas.

Next, we need to create a page that can focus on a single topic, showing the topic name and all the entries for that topic. We'll define a new URL pattern, write a view, and create a template. We'll also modify the topics page so each item in the bulleted list links to its corresponding topic page.

[El patrón URL de temas \(The Topic URL Pattern\)](#)

El patrón URL para la página de temas es un poco diferente de los patrones URL anteriores porque utilizará el atributo `id` del tema para indicar qué tema se ha solicitado. Por ejemplo, si el usuario quiere ver la página detallada del tema Ajedrez (donde `id` es 1), la URL será `http://localhost:8000/topics/1/`. Aquí tienes un patrón para que coincida con esta URL, que debes colocar en `learning_logs/urls.py`:

The URL pattern for the topic page is a little different from the prior URL patterns because it will use the topic's `id` attribute to indicate which topic was requested. For example, if the user wants to see the detail page for the Chess topic (where the `id` is 1), the URL will be

`http://localhost:8000/topics/1/` . Here's a pattern to match this URL, which you should place in `learning_logs/urls.py` :

learning_logs/urls.py

learning_logs/urls.py

```
--snip--
urlpatterns = [
    --snip--
    # Detail page for a single topic.
    path('topics/<int:topic_id>/', views.topic, name='topic'),
]
```

Examinemos la cadena `'topics/<int:topic_id>/'` en este patrón de URL. La primera parte de la cadena indica a Django que busque URLs que tengan la palabra *topics* después de la URL base. La segunda parte de la cadena, `/<int:topic_id>/` , busca un número entero entre dos barras diagonales y asigna el valor entero a un argumento llamado `topic_id` .

Let's examine the string `'topics/<int:topic_id>/'` in this URL pattern. The first part of the string tells Django to look for URLs that have the word *topics* after the base URL. The second part of the string, `/<int:topic_id>/` , matches an integer between two forward slashes and assigns the integer value to an argument called `topic_id` .

Cuando Django encuentra una URL que coincide con este patrón, llama a la función de vista `topic()` con el valor asignado a `topic_id` como argumento. Utilizaremos el valor de `topic_id` para obtener el tema correcto dentro de la función.

When Django finds a URL that matches this pattern, it calls the view function `topic()` with the value assigned to `topic_id` as an argument. We'll use the value of `topic_id` to get the correct topic inside the function.

La vista Tema (The Topic View)

La función `topic()` necesita obtener el tema y todas las entradas asociadas de la base de datos, de forma muy similar a lo que hicimos anteriormente en el shell de Django:

The `topic()` function needs to get the topic and all associated entries from the database, much like what we did earlier in the Django shell:

views.py

views.py

--snip--

```
❶ def topic(request, topic_id):  
    """Show a single topic and all its entries."""  
    ❷ topic = Topic.objects.get(id=topic_id)  
    ❸ entries = topic.entry_set.order_by('-date_added')  
    ❹ context = {'topic': topic, 'entries': entries}  
    ❺ return render(request, 'learning_logs/topic.xhtml', context)
```

Esta es la primera función de vista que requiere un parámetro distinto del objeto `request`. La función acepta el valor capturado por la expresión `<int:topic_id>/` y lo asigna a `topic_id` ❶. A continuación, utilizamos `get()` para recuperar el tema, tal y como hicimos en el shell de Django ❷. A continuación, obtenemos todas las entradas asociadas a este tema y las ordenamos según `date_added` ❸. El signo menos delante de `date_added` ordena los resultados en orden inverso, lo que mostrará primero las entradas más recientes. Almacenamos el tema y las entradas en el diccionario `context` ❹ y llamamos a `render()` con el objeto `request`, la plantilla `topic.xhtml` y el diccionario `context` ❺.

This is the first view function that requires a parameter other than the `request` object. The function accepts the value captured by the expression `<int:topic_id>/` and assigns it to `topic_id` ❶. Then we use `get()` to retrieve the topic, just as we did in the Django shell ❷. Next, we get all of the entries associated with this topic and order them according to `date_added` ❸. The minus sign in front of `date_added` sorts the results in reverse order, which will display the most recent entries first. We store the topic and entries in the context dictionary ❹ and call `render()` with the `request` object, the `topic.xhtml` template, and the context dictionary ❺.

Nota (Note)

Las frases de código en ❷ y ❸ se llaman *queries*, porque consultan la base de datos en busca de información específica. Cuando escribas consultas como éstas en tus propios proyectos, es útil probarlas primero en el shell de Django. Obtendrás una respuesta mucho más rápida en el intérprete de comandos que escribiendo una vista y una plantilla y comprobando después los resultados en un navegador.

The code phrases at ❷ and ❸ are called *queries*, because they query the database for specific information. When you're writing queries like these in your own projects, it's helpful to try them out in the Django shell first. You'll get much quicker feedback in the shell than you would by writing a view and template and then checking the results in a browser.

La plantilla de temas (The Topic Template)

La plantilla necesita mostrar el nombre del tema y las entradas. También necesitamos informar al usuario si aún no se han realizado entradas para este tema.

The template needs to display the name of the topic and the entries. We also need to inform the user if no entries have been made yet for this topic.

tema.xhtml

topic.xhtml

```
{% extends 'learning_logs/base.xhtml' %}
```

```
{% block content %}
```

```
❶ <p>Topic: {{ topic.text }}</p>
```

```

<p>Entries:</p>
❷ <ul>
❸   {% for entry in entries %}
      <li>
❹     <p>{{ entry.date_added|date:'M d, Y H:i' }}</p>
❺     <p>{{ entry.text|linebreaks }}</p>
      </li>
❻   {% empty %}
      <li>There are no entries for this topic yet.</li>
    {% endfor %}
  </ul>

{% endblock content %}

```

Extendemos *base.xhtml* , como haremos para todas las páginas del proyecto. A continuación, mostramos el atributo `text` del tema que se ha solicitado ❶ . La variable `topic` está disponible porque está incluida en el diccionario `context` . A continuación, iniciamos una lista con viñetas ❷ para mostrar cada una de las entradas y hacemos un bucle a través de ellas ❸ , como hicimos antes con los temas.

We extend *base.xhtml* , as we'll do for all pages in the project. Next, we show the `text` attribute of the topic that's been requested ❶ . The variable `topic` is available because it's included in the `context` dictionary. We then start a bulleted list ❷ to show each of the entries and loop through them ❸ , as we did with the topics earlier.

Cada viñeta contiene dos datos: la marca de tiempo y el texto completo de cada entrada. Para la fecha y hora ❹ , mostramos el valor del atributo `date_added` . En las plantillas Django, una línea vertical (`|`) representa una plantilla *filter* -una función que modifica el valor de una variable de plantilla durante el proceso de representación. El filtro `date:'M d, Y H:i'` muestra marcas de tiempo en el formato *January 1, 2022 23:00* . La siguiente línea muestra el valor del atributo `text` de la entrada actual. El filtro `linebreaks` ❺ garantiza que las entradas de texto largas incluyan saltos de línea en un formato que entiendan los navegadores, en lugar de mostrar un bloque de texto ininterrumpido. Volvemos a utilizar la etiqueta de plantilla `{% empty %}` ❻ para imprimir un mensaje que informe al usuario de que no se han realizado entradas.

Each bullet lists two pieces of information: the timestamp and the full text of each entry. For the timestamp ❹ , we display the value of the attribute `date_added` . In Django templates, a vertical line (`|`) represents a template *filter* —a function that modifies the value in a template variable during the rendering process. The filter `date:'M d, Y H:i'`

displays timestamps in the format *January 1, 2022 23:00* . The next line displays the value of the current entry's `text` attribute. The filter `linebreaks` ⑤ ensures that long text entries include line breaks in a format understood by browsers, rather than showing a block of uninterrupted text. We again use the `{% empty %}` template tag ⑥ to print a message informing the user that no entries have been made.

Enlaces desde la Página de Temas (Links from the Topics Page)

Antes de ver la página de temas en un navegador, tenemos que modificar la plantilla de temas para que cada tema enlace con la página adecuada. Este es el cambio que tienes que hacer en *topics.xhtml* :

Before we look at the topic page in a browser, we need to modify the topics template so each topic links to the appropriate page. Here's the change you need to make to *topics.xhtml* :

temas.xhtml

topics.xhtml

```
--snip--
{% for topic in topics %}
  <li>
    <a href="{% url 'learning_logs:topic' topic.id %}">
      {{ topic.text }}</a></li>
  </li>
{% empty %}
--snip--
```

Utilizamos la etiqueta de plantilla de URL para generar el enlace adecuado, basándonos en el patrón de URL de `learning_logs` con el nombre `'topic'` . Este patrón de URL requiere un argumento `topic_id` , por lo que añadimos el atributo `topic.id` a la etiqueta de plantilla de URL. Ahora cada tema de la lista de temas es un enlace a una página de tema, como *http://localhost:8000/topics/1/* .

We use the URL template tag to generate the proper link, based on the URL pattern in `learning_logs` with the name `'topic'`. This URL pattern requires a `topic_id` argument, so we add the attribute `topic.id` to the URL template tag. Now each topic in the list of topics is a link to a topic page, such as `http://localhost:8000/topics/1/`.

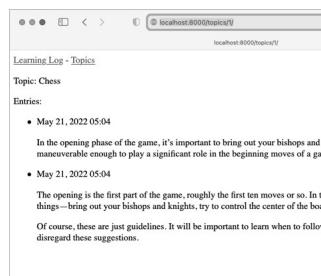
Cuando actualices la página de temas y hagas clic en un tema, deberías ver una página parecida a [la Figura 18-5](#).

When you refresh the topics page and click a topic, you should see a page that looks like [Figure 18-5](#).

Nota (Note)

Hay una sutil pero importante diferencia entre `topic.id` y `topic_id`. La expresión `topic.id` examina un tema y recupera el valor del ID correspondiente. La variable `topic_id` es una referencia a ese ID en el código. Si te encuentras con errores al trabajar con IDs, asegúrate de que estás utilizando estas expresiones de la forma adecuada.

There's a subtle but important difference between `topic.id` and `topic_id`. The expression `topic.id` examines a topic and retrieves the value of the corresponding ID. The variable `topic_id` is a reference to that ID in the code. If you run into errors when working with IDs, make sure you're using these expressions in the appropriate ways.



DESCRIPCIÓN DE LA
IMAGEN (En español):

Registro de
Aprendizaje - Temas
Tema: Ajedrez
Entradas: - 21 de mayo
de 2022 05:04 En la
fase inicial del juego, es
importante desarrollar
tus alfiles y caballos.
Estas piezas son
poderosas y

maniobrables, lo que les permite desempeñar un papel importante en los movimientos iniciales de una partida. - 21 de mayo de 2022 05:04 La apertura es la primera parte del juego, por lo que intenta controlar los primeros diez movimientos o algo así. En la apertura, es buena idea hacer tres cosas: sacar tus alfiles y caballos, intentar controlar el centro del tablero y enrocar a tu rey. Por supuesto, estas son solo pautas. Es importante aprender cuándo seguir estas pautas y cuándo ignorar estas sugerencias.

Figura 18-5 : Página detallada de un tema, mostrando todas las entradas de un tema

Figure 18-5 : The detail page for a single topic, showing all entries for a topic

Pruébalo tú mismo (Try It Yourself)

18-7. Documentación de plantillas: Hojea la documentación sobre plantillas de Django en <https://docs.djangoproject.com/en/4.1/ref/templates> . Puedes consultarla cuando trabajes en tus propios proyectos.

18-7. Template Documentation: Skim the Django template documentation at <https://docs.djangoproject.com/en/4.1/ref/templates> . You can refer back to it when you're working on your own projects.

18-8. Páginas de la Pizzería: Añade una página al proyecto Pizzería del Ejercicio 18-6 (página 392) que muestre los nombres de las pizzas disponibles. Luego enlaza cada

nombre de pizza a una página que muestre los ingredientes de la pizza. Asegúrate de utilizar la herencia de plantillas para construir tus páginas de forma eficiente.

18-8. Pizzeria Pages: Add a page to the Pizzeria project from Exercise 18-6 (page 392) that shows the names of available pizzas. Then link each pizza name to a page displaying the pizza's toppings. Make sure you use template inheritance to build your pages efficiently.

Resumen (Summary)

En este capítulo, has aprendido cómo empezar a construir una aplicación web sencilla utilizando el framework Django. Has visto una breve especificación del proyecto, has instalado Django en un entorno virtual, has configurado un proyecto y has comprobado que el proyecto estaba configurado correctamente. Configuraste una aplicación y definiste modelos para representar los datos de tu aplicación. Aprendiste sobre bases de datos y cómo Django te ayuda a migrar tu base de datos después de hacer un cambio en tus modelos. Creaste un superusuario para el sitio de administración y lo utilizaste para introducir algunos datos iniciales.

In this chapter, you learned how to start building a simple web app using the Django framework. You saw a brief project specification, installed Django to a virtual environment, set up a project, and checked that the project was set up correctly. You set up an app and defined models to represent the data for your app. You learned about databases and how Django helps you migrate your database after you make a change to your models. You created a superuser for the admin site, and you used the admin site to enter some initial data.

También exploraste el shell de Django, que te permite trabajar con los datos de tu proyecto en una sesión de terminal. Aprendiste a definir URLs, crear funciones de vista y escribir plantillas para crear páginas para tu sitio. También utilizaste la herencia de plantillas para simplificar la estructura de las plantillas individuales y facilitar la modificación del sitio a medida que evoluciona el proyecto.

You also explored the Django shell, which allows you to work with your project's data in a terminal session. You learned how to define URLs, create view functions, and write templates to make pages for your site. You also used template inheritance to simplify the

structure of individual templates and make it easier to modify the site as the project evolves.

En el Capítulo 19 , crearás páginas intuitivas y fáciles de usar que permitan a los usuarios añadir nuevos temas y entradas y editar las entradas existentes sin pasar por el sitio de administración. También añadirás un sistema de registro de usuarios, que les permita crear una cuenta y crear su propio registro de aprendizaje. Este es el corazón de una aplicación web: la capacidad de crear algo con lo que pueda interactuar cualquier número de usuarios.

In Chapter 19 , you'll make intuitive, user-friendly pages that allow users to add new topics and entries and edit existing entries without going through the admin site. You'll also add a user registration system, allowing users to create an account and make their own learning log. This is the heart of a web app—the ability to create something that any number of users can interact with.

Capítulo 19: Cuentas de usuario (Chapter 19: User Accounts)

19

Cuentas de usuario (19 User Accounts)



DESCRIPCIÓN DE LA
IMAGEN (En español):

La imagen muestra un dibujo en blanco y negro de una serpiente con gafas, que está envuelta alrededor de una turbina o motor a reacción. La expresión de la serpiente parece decidida o enfocada, y el diseño tiene un estilo caricaturesco. El fondo es circular y de color gris oscuro.

En el corazón de una aplicación web está la posibilidad de que cualquier usuario, en cualquier parte del mundo, pueda registrar una cuenta en tu aplicación y empezar a utilizarla. En este capítulo, construirás formularios para que los usuarios puedan añadir sus propios temas y entradas, y editar las entradas existentes. También aprenderás cómo Django protege contra ataques comunes a las páginas basadas en formularios, para que no tengas que dedicar mucho tiempo a pensar en la seguridad de tus aplicaciones.

At the heart of a web application is the ability for any user, anywhere in the world, to register an account with your app and start using it. In this chapter, you'll build forms so users can add their own topics and entries, and edit existing entries. You'll also learn how Django guards against common attacks against form-based pages, so you won't have to spend much time thinking about securing your apps.

También implementarás un sistema de autenticación de usuarios. Construirás una página de registro para que los usuarios creen cuentas, y luego restringirás el acceso a ciertas páginas sólo a los usuarios registrados. Luego modificarás algunas de las funciones de vista para que los usuarios sólo puedan ver sus propios datos. Aprenderás a mantener seguros los datos de tus usuarios.

You'll also implement a user authentication system. You'll build a registration page for users to create accounts, and then restrict access to certain pages to logged-in users only. Then you'll modify some of the view functions so users can only see their own data. You'll learn to keep your users' data safe and secure.

Permitir que los usuarios introduzcan datos *(Allowing Users to Enter Data)*

Antes de construir un sistema de autenticación para crear cuentas, primero añadiremos algunas páginas que permitan a los usuarios introducir sus propios datos. Daremos a los usuarios la posibilidad de añadir un nuevo tema, añadir una nueva entrada y editar sus entradas anteriores.

Before we build an authentication system for creating accounts, we'll first add some pages that allow users to enter their own data. We'll give users the ability to add a new topic, add a new entry, and edit their previous entries.

Actualmente, sólo un superusuario puede introducir datos a través del sitio de administración. No queremos que los usuarios interactúen con el sitio de administración, así que utilizaremos las herramientas de creación de formularios de Django para crear páginas que permitan a los usuarios introducir datos.

Currently, only a superuser can enter data through the admin site. We don't want users to interact with the admin site, so we'll use Django's form-building tools to build pages that allow users to enter data.

Añadir nuevos temas *(Adding New Topics)*

Empecemos por permitir a los usuarios añadir un nuevo tema. Añadir una página basada en un formulario funciona de forma muy parecida a añadir las páginas que ya hemos construido: definimos una URL, escribimos una función de vista y escribimos una plantilla. La única diferencia significativa es la adición de un nuevo módulo llamado *forms.py*, que contendrá los formularios.

Let's start by allowing users to add a new topic. Adding a form-based page works in much the same way as adding the pages we've already built: we define a URL, write a view function, and write a template. The one significant difference is the addition of a new module called *forms.py*, which will contain the forms.

El Tema ModeloFormulario (The Topic ModelForm)

Cualquier página que permita a un usuario introducir y enviar información en una página web implica un elemento HTML llamado *form* . Cuando los usuarios introducen información, necesitamos *validate* que la información proporcionada sea el tipo correcto de datos y no sea maliciosa, como código diseñado para interrumpir nuestro servidor. A continuación, necesitamos procesar y guardar la información válida en el lugar adecuado de la base de datos. Django automatiza gran parte de este trabajo.

Any page that lets a user enter and submit information on a web page involves an HTML element called a *form* . When users enter information, we need to *validate* that the information provided is the right kind of data and is not malicious, such as code designed to interrupt our server. We then need to process and save valid information to the appropriate place in the database. Django automates much of this work.

La forma más sencilla de construir un formulario en Django es utilizar un `ModelForm` , que utiliza la información de los modelos que definimos en el Capítulo 18 para construir un formulario automáticamente. Escribe tu primer formulario en el archivo *forms.py* , que debe crearse en el mismo directorio que *models.py* :

The simplest way to build a form in Django is to use a `ModelForm` , which uses the information from the models we defined in Chapter 18 to build a form automatically. Write your first form in the file *forms.py* , which should be created in the same directory as *models.py* :

formularios.py

forms.py

```
from django import forms
```

```
from .models import Topic
```

```
❶ class TopicForm(forms.ModelForm):  
    class Meta:
```

```
❷    model = Topic
❸    fields = ['text']
❹    labels = {'text': ''}
```

Primero importamos el módulo `forms` y el modelo con el que trabajaremos, `Topic`. A continuación, definimos una clase llamada `TopicForm`, que hereda de `forms.ModelForm` ❶.

We first import the `forms` module and the model we'll work with, `Topic`. We then define a class called `TopicForm`, which inherits from `forms.ModelForm` ❶.

La versión más sencilla de `ModelForm` consiste en una clase anidada `Meta` que indica a Django en qué modelo debe basarse el formulario y qué campos debe incluir en él. Aquí especificamos que el formulario debe basarse en el modelo `Topic` ❷, y que sólo debe incluir el campo `text` ❸. La cadena vacía en el diccionario de etiquetas indica a Django que no genere una etiqueta para el campo `text` ❹.

The simplest version of a `ModelForm` consists of a nested `Meta` class telling Django which model to base the form on and which fields to include in the form. Here we specify that the form should be based on the `Topic` model ❷, and that it should only include the `text` field ❸. The empty string in the labels dictionary tells Django not to generate a label for the `text` field ❹.

La URL del nuevo_tema (The new_topic URL)

La URL de una nueva página debe ser corta y descriptiva. Cuando el usuario quiera añadir un nuevo tema, le enviaremos a `http://localhost:8000/new_topic/`. Éste es el patrón de URL para la página `new_topic`; añádelo a `learning_logs/urls.py`:

The URL for a new page should be short and descriptive. When the user wants to add a new topic, we'll send them to `http://localhost:8000/new_topic/`. Here's the URL pattern for the `new_topic` page; add this to `learning_logs/urls.py`:

learning_logs/urls.py

learning_logs/urls.py

```
--snip--
urlpatterns = [
    --snip--
    # Page for adding a new topic.
    path('new_topic/', views.new_topic, name='new_topic'),
]
```

Este patrón de URL envía solicitudes a la función de vista `new_topic()` , que escribiremos a continuación.

This URL pattern sends requests to the view function `new_topic()` , which we'll write next.

La función de vista `new_topic()` (The `new_topic()` View Function)

La función `new_topic()` debe gestionar dos situaciones diferentes: las solicitudes iniciales de la página `new_topic` , en cuyo caso debe mostrar un formulario en blanco; y el procesamiento de los datos enviados en el formulario. Una vez procesados los datos de un formulario enviado, debe redirigir al usuario de nuevo a la página `topics` :

The `new_topic()` function needs to handle two different situations: initial requests for the `new_topic` page, in which case it should show a blank form; and the processing of any data submitted in the form. After data from a submitted form is processed, it needs to redirect the user back to the `topics` page:

views.py

views.py

```
from django.shortcuts import render, redirect
```

```
from .models import Topic
from .forms import TopicForm
```

```
--snip--
def new_topic(request):
    """Add a new topic."""
    ❶ if request.method != 'POST':
        # No data submitted; create a blank form.
    ❷     form = TopicForm()
    else:
        # POST data submitted; process data.
    ❸     form = TopicForm(data=request.POST)
    ❹     if form.is_valid():
    ❺         form.save()
    ❻     return redirect('learning_logs:topics')

    # Display a blank or invalid form.
    ❼     context = {'form': form}
    return render(request, 'learning_logs/new_topic.xhtml', context)
```

Importamos la función `redirect` , que utilizaremos para redirigir al usuario a la página `topics` después de que envíe su tema. También importamos el formulario que acabamos de escribir, `TopicForm` .

We import the function `redirect` , which we'll use to redirect the user back to the `topics` page after they submit their topic. We also import the form we just wrote, `TopicForm` .

Peticiones GET y POST (GET and POST Requests)

Los dos tipos principales de peticiones que utilizarás cuando construyas aplicaciones son GET y POST. Las peticiones *GET* se utilizan para páginas que sólo leen datos del servidor. Normalmente utilizas las peticiones *POST* cuando el usuario necesita enviar información a través de un formulario. Especificaremos el método POST para procesar todos nuestros formularios. (Existen algunos otros tipos de peticiones, pero no los utilizaremos en este proyecto).

The two main types of requests you'll use when building apps are GET and POST. You use *GET* requests for pages that only read data from the server. You usually use *POST* requests when the user needs to submit information through a form. We'll be specifying the POST method for processing all of our forms. (A few other kinds of requests exist, but we won't use them in this project.)

La función `new_topic()` toma como parámetro el objeto `request` . Cuando el usuario solicite inicialmente esta página, su navegador enviará una solicitud GET. Una vez que el usuario haya rellenado y enviado el formulario, su navegador enviará una solicitud POST. Dependiendo de la petición, sabremos si el usuario está solicitando un formulario en blanco (GET) o pidiéndonos que procesemos un formulario cumplimentado (POST).

The `new_topic()` function takes in the `request` object as a parameter. When the user initially requests this page, their browser will send a GET request. Once the user has filled out and submitted the form, their browser will submit a POST request. Depending on the request, we'll know whether the user is requesting a blank form (GET) or asking us to process a completed form (POST).

Utilizamos una prueba `if` para determinar si el método de solicitud es GET o POST ❶ . Si el método de solicitud no es POST, probablemente la solicitud sea GET, por lo que tenemos que devolver un formulario en blanco. (Si es otro tipo de solicitud, sigue siendo seguro devolver un formulario en blanco.) Creamos una instancia de `TopicForm` ❷ , la asignamos a la variable `form` , y enviamos el formulario a la plantilla del diccionario `context` ❸ . Como no incluimos argumentos al instanciar `TopicForm` , Django crea un formulario en blanco que el usuario puede rellenar.

We use an `if` test to determine whether the request method is GET or POST ❶ . If the request method isn't POST, the request is probably GET, so we need to return a blank form. (If it's another kind of request, it's still safe to return a blank form.) We make an instance of `TopicForm` ❷ , assign it to the variable `form` , and send the form to the template in the `context` dictionary ❸ . Because we included no arguments when instantiating `TopicForm` , Django creates a blank form that the user can fill out.

Si el método de solicitud es POST, el bloque `else` se ejecuta y procesa los datos enviados en el formulario. Creamos una instancia de `TopicForm` ❹ y le pasamos los datos introducidos por el usuario, que se asignan a `request.POST` . El objeto `form` que se devuelve contiene la información enviada por el usuario.

If the request method is POST, the `else` block runs and processes the data submitted in the form. We make an instance of `TopicForm` ❹ and pass it the data entered by the user, which is assigned to `request.POST` . The `form` object that's returned contains the information submitted by the user.

No podemos guardar la información enviada en la base de datos hasta que hayamos comprobado que es válida ④ . El método `is_valid()` comprueba que se han rellenado todos los campos obligatorios (todos los campos de un formulario son obligatorios por defecto) y que los datos introducidos coinciden con los tipos de campo esperados; por ejemplo, que la longitud de `text` es inferior a 200 caracteres, como especificamos en *models.py* en el Capítulo 18 . Esta validación automática nos ahorra mucho trabajo. Si todo es válido, podemos llamar a `save()` ⑤ , que escribe los datos del formulario en la base de datos.

We can't save the submitted information in the database until we've checked that it's valid ④ . The `is_valid()` method checks that all required fields have been filled in (all fields in a form are required by default) and that the data entered matches the field types expected—for example, that the length of `text` is less than 200 characters, as we specified in *models.py* in Chapter 18 . This automatic validation saves us a lot of work. If everything is valid, we can call `save()` ⑤ , which writes the data from the form to the database.

Una vez guardados los datos, podemos abandonar esta página. La función `redirect()` toma el nombre de una vista y redirige al usuario a la página asociada a esa vista. Aquí utilizamos `redirect()` para redirigir el navegador del usuario a la página `topics` ⑥ , donde el usuario debería ver el tema que acaba de introducir en la lista de temas.

Once we've saved the data, we can leave this page. The `redirect()` function takes in the name of a view and redirects the user to the page associated with that view. Here we use `redirect()` to redirect the user's browser to the `topics` page ⑥ , where the user should see the topic they just entered in the list of topics.

La variable `context` se define al final de la función de la vista, y la página se renderiza utilizando la plantilla *new_topic.xhtml* , que crearemos a continuación. Este código se coloca fuera de cualquier bloque `if` ; se ejecutará si se ha creado un formulario en blanco, y se ejecutará si se determina que un formulario enviado no es válido. Un formulario inválido incluirá algunos mensajes de error por defecto para ayudar al usuario a enviar datos aceptables.

The `context` variable is defined at the end of the view function, and the page is rendered using the template *new_topic.xhtml* , which we'll create next. This code is placed outside of

any if block; it will run if a blank form was created, and it will run if a submitted form is determined to be invalid. An invalid form will include some default error messages to help the user submit acceptable data.

La plantilla new_topic (The new_topic Template)

Ahora crearemos una nueva plantilla llamada *new_topic.xhtml* para mostrar el formulario que acabamos de crear:

Now we'll make a new template called *new_topic.xhtml* to display the form we just created:

nuevo_tema.xhtml

new_topic.xhtml

```
{% extends "learning_logs/base.xhtml" %}

{% block content %}
    <p>Add a new topic:</p>

    ❶ <form action="{% url 'learning_logs:new_topic' %}" method='post'>
    ❷   {% csrf_token %}
    ❸   {{ form.as_div }}
    ❹   <button name="submit">Add topic</button>
    </form>

{% endblock content %}
```

Esta plantilla extiende *base.xhtml*, por lo que tiene la misma estructura base que el resto de páginas del Registro de Aprendizaje. Utilizamos las etiquetas `<form></form>` para definir un formulario HTML ❶. El argumento `action` indica al navegador dónde enviar los datos enviados en el formulario; en este caso, los enviamos a la función de vista `new_topic()`. El argumento `method` indica al navegador que envíe los datos como una petición POST.

This template extends *base.xhtml*, so it has the same base structure as the rest of the pages in Learning Log. We use the `<form></form>` tags to define an HTML form ❶. The `action`

argument tells the browser where to send the data submitted in the form; in this case, we send it back to the view function `new_topic()`. The `method` argument tells the browser to submit the data as a POST request.

Django utiliza la etiqueta de plantilla `{% csrf_token %}` ❷ para evitar que los atacantes utilicen el formulario para obtener acceso no autorizado al servidor. (Este tipo de ataque se denomina *cross-site request forgery*.) A continuación, mostramos el formulario; aquí puedes ver lo sencillas que Django puede hacer ciertas tareas, como mostrar un formulario. Sólo tenemos que incluir la variable de plantilla `{{ form.as_div }}` para que Django cree automáticamente todos los campos necesarios para mostrar el formulario ❸. El modificador `as_div` indica a Django que muestre todos los elementos del formulario como elementos HTML `<div></div>`; ésta es una forma sencilla de mostrar el formulario de forma ordenada.

Django uses the template tag `{% csrf_token %}` ❷ to prevent attackers from using the form to gain unauthorized access to the server. (This kind of attack is called a *cross-site request forgery*.) Next, we display the form; here you can see how simple Django can make certain tasks, such as displaying a form. We only need to include the template variable `{{ form.as_div }}` for Django to create all the fields necessary to display the form automatically ❸. The `as_div` modifier tells Django to render all the form elements as HTML `<div></div>` elements; this is a simple way to display the form neatly.

Django no crea un botón de envío para los formularios, así que definimos uno antes de cerrar el formulario ❹.

Django doesn't create a submit button for forms, so we define one before closing the form ❹.

[Enlace a la página nuevo_tema \(Linking to the new_topic Page\)](#)

A continuación, incluimos un enlace a la página `new_topic` en la página `topics`:

Next, we include a link to the `new_topic` page on the `topics` page:

temas.xhtml

topics.xhtml

```
{% extends "learning_logs/base.xhtml" %}
```

```
{% block content %}
```

```
<p>Topics</p>
```

```
<ul>
```

```
--snip--
```

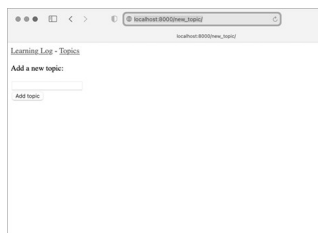
```
</ul>
```

```
<a href="{% url 'learning_logs:new_topic' %}">Add a new topic</a>
```

```
{% endblock content %}
```

Coloca el enlace después de la lista de temas existentes. [La Figura 19-1](#) muestra el formulario resultante; prueba a utilizarlo para añadir algunos temas nuevos por tu cuenta.

Place the link after the list of existing topics. [Figure 19-1](#) shows the resulting form; try using the form to add a few new topics of your own.



DESCRIPCIÓN DE LA
IMAGEN (En español):

Registro de
Aprendizaje - Temas
Añadir un nuevo tema:
Añadir tema

Figura 19-1 : Página para añadir un nuevo tema

Figure 19-1 : The page for adding a new topic

Añadir nuevos temas (Adding New Entries)

Ahora que el usuario puede añadir un nuevo tema, también querrá añadir nuevas entradas. Definiremos de nuevo una URL, escribiremos una función de vista y una plantilla, y enlazaremos con la página. Pero antes, añadiremos otra clase a *forms.py*.

Now that the user can add a new topic, they'll want to add new entries too. We'll again define a URL, write a view function and a template, and link to the page. But first, we'll add another class to *forms.py*.

El modelo de formulario de entrada (The Entry ModelForm)

Necesitamos crear un formulario asociado al modelo `Entry`, pero esta vez, con un poco más de personalización que `TopicForm`:

We need to create a form associated with the `Entry` model, but this time, with a bit more customization than `TopicForm`:

formularios.py

forms.py

```
from django import forms
```

```
from .models import Topic, Entry
```

```
class TopicForm(forms.ModelForm):  
    --snip--
```

```
class EntryForm(forms.ModelForm):  
    class Meta:  
        model = Entry  
        fields = ['text']  
❶ labels = {'text': ''}  
❷ widgets = {'text': forms.Textarea(attrs={'cols': 80})}
```

Actualizamos la declaración `import` para incluir tanto `Entry` como `Topic`. Creamos una nueva clase llamada `EntryForm` que hereda de `forms.ModelForm`. La clase `EntryForm` tiene una clase anidada `Meta` que enumera el modelo en el que se basa y el campo a incluir en el formulario. Volvemos a dar al campo `'text'` una etiqueta en blanco ❶.

We update the `import` statement to include `Entry` as well as `Topic`. We make a new class called `EntryForm` that inherits from `forms.ModelForm`. The `EntryForm` class has a nested `Meta` class listing the model it's based on, and the field to include in the form. We again give the field `'text'` a blank label ❶.

Para `EntryForm`, incluimos el atributo `widgets` ❷. Un *widget* es un elemento de formulario HTML, como un cuadro de texto de una sola línea, un área de texto de varias líneas o una lista desplegable. Al incluir el atributo `widgets`, puedes anular las opciones predeterminadas de `widgets` de Django. Aquí le estamos diciendo a Django que utilice un elemento `forms.Textarea` con una anchura de 80 columnas, en lugar de las 40 columnas por defecto. Esto da a los usuarios espacio suficiente para escribir una entrada significativa.

For `EntryForm`, we include the `widgets` attribute ❷. A *widget* is an HTML form element, such as a single-line text box, multiline text area, or drop-down list. By including the `widgets` attribute, you can override Django's default widget choices. Here we're telling Django to use a `forms.Textarea` element with a width of 80 columns, instead of the default 40 columns. This gives users enough room to write a meaningful entry.

La URL de la nueva_entrada (The new_entry URL)

Las nuevas entradas deben asociarse a un tema concreto, por lo que necesitamos incluir un argumento `topic_id` en la URL para añadir una nueva entrada. Aquí tienes la URL, que debes añadir a *learning_logs/urls.py*:

New entries must be associated with a particular topic, so we need to include a `topic_id` argument in the URL for adding a new entry. Here's the URL, which you add to *learning_logs/urls.py*:

learning_logs/urls.py

learning_logs/urls.py

```
--snip--
urlpatterns = [
    --snip--
    # Page for adding a new entry.
    path('new_entry/<int:topic_id>/', views.new_entry, name='new_entry'),
]
```

Este patrón de URL coincide con cualquier URL de la forma `http://localhost:8000/new_entry/id/`, donde `id` es un número que coincide con el ID del tema. El código `<int:topic_id>` captura un valor numérico y lo asigna a la variable `topic_id`. Cuando se solicita una URL que coincide con este patrón, Django envía la solicitud y el ID del tema a la función de vista `new_entry()`.

This URL pattern matches any URL with the form `http://localhost:8000/new_entry/id/`, where `id` is a number matching the topic ID. The code `<int:topic_id>` captures a numerical value and assigns it to the variable `topic_id`. When a URL matching this pattern is requested, Django sends the request and the topic's ID to the `new_entry()` view function.

La función de vista `new_entry()` (The `new_entry()` View Function)

La función de vista para `new_entry` es muy parecida a la función para añadir un nuevo tema. Añade el siguiente código a tu archivo `views.py`:

The view function for `new_entry` is much like the function for adding a new topic. Add the following code to your `views.py` file:

vistas.py

views.py

```
from django.shortcuts import render, redirect
```

```
from .models import Topic
```

```
from .forms import TopicForm, EntryForm
```

```
--snip--
```



```

def new_entry(request, topic_id):
    """Add a new entry for a particular topic."""
    ❶ topic = Topic.objects.get(id=topic_id)

    ❷ if request.method != 'POST':
        # No data submitted; create a blank form.
    ❸ form = EntryForm()
    else:
        # POST data submitted; process data.
    ❹ form = EntryForm(data=request.POST)
        if form.is_valid():
    ❺ new_entry = form.save(commit=False)
    ❻ new_entry.topic = topic
        new_entry.save()
    ❼ return redirect('learning_logs:topic', topic_id=topic_id)

    # Display a blank or invalid form.
    context = {'topic': topic, 'form': form}
    return render(request, 'learning_logs/new_entry.xhtml', context)

```

Actualizamos la declaración `import` para incluir el `EntryForm` que acabamos de hacer. La definición de `new_entry()` tiene un parámetro `topic_id` para almacenar el valor que recibe de la URL. Necesitaremos el tema para renderizar la página y procesar los datos del formulario, así que utilizamos `topic_id` para obtener el objeto tema correcto ❶ .

We update the `import` statement to include the `EntryForm` we just made. The definition of `new_entry()` has a `topic_id` parameter to store the value it receives from the URL. We'll need the topic to render the page and process the form's data, so we use `topic_id` to get the correct topic object ❶ .

A continuación, comprobamos si el método de solicitud es POST o GET ❷ . El bloque `if` se ejecuta si se trata de una solicitud GET, y creamos una instancia en blanco de `EntryForm` ❸ .

Next, we check whether the request method is POST or GET ❷ . The `if` block executes if it's a GET request, and we create a blank instance of `EntryForm` ❸ .

Si el método de solicitud es POST, procesamos los datos creando una instancia de `EntryForm` , rellena con los datos POST del objeto `request` ④ . A continuación, comprobamos si el formulario es válido. Si lo es, tenemos que establecer el atributo `topic` del objeto de entrada antes de guardarlo en la base de datos. Cuando llamamos a `save()` , incluimos el argumento `commit=False` ⑤ para decirle a Django que cree un nuevo objeto de entrada y lo asigne a `new_entry` , sin guardarlo aún en la base de datos. Establecemos el atributo `topic` de `new_entry` al tema que sacamos de la base de datos al principio de la función ⑥ . A continuación, llamamos a `save()` sin argumentos, guardando la entrada en la base de datos con el tema asociado correcto.

If the request method is POST, we process the data by making an instance of `EntryForm` , populated with the POST data from the `request` object ④ . We then check whether the form is valid. If it is, we need to set the entry object's `topic` attribute before saving it to the database. When we call `save()` , we include the argument `commit=False` ⑤ to tell Django to create a new entry object and assign it to `new_entry` , without saving it to the database yet. We set the `topic` attribute of `new_entry` to the topic we pulled from the database at the beginning of the function ⑥ . Then we call `save()` with no arguments, saving the entry to the database with the correct associated topic.

La llamada a `redirect()` requiere dos argumentos: el nombre de la vista a la que queremos redirigir y el argumento que requiere la función ⑦ . En este caso, estamos redirigiendo a `topic()` , que necesita el argumento `topic_id` . Esta vista muestra la página del tema para el que el usuario hizo una entrada, y debería ver su nueva entrada en la lista de entradas.

The `redirect()` call requires two arguments: the name of the view we want to redirect to and the argument that view function requires ⑦ . Here, we're redirecting to `topic()` , which needs the argument `topic_id` . This view then renders the topic page that the user made an entry for, and they should see their new entry in the list of entries.

Al final de la función, creamos un diccionario `context` y mostramos la página utilizando la plantilla `new_entry.xhtml` . Este código se ejecutará para un formulario en blanco, o para un formulario que ha sido enviado pero resulta ser inválido.

At the end of the function, we create a `context` dictionary and render the page using the `new_entry.xhtml` template. This code will execute for a blank form, or for a form that's been submitted but turns out to be invalid.

La plantilla new_entry (The new_entry Template)

Como puedes ver en el código siguiente, la plantilla para `new_entry` es similar a la plantilla para `new_topic` :

As you can see in the following code, the template for `new_entry` is similar to the template for `new_topic` :

nueva_entrada.xhtml

new_entry.xhtml

```
{% extends "learning_logs/base.xhtml" %}
```

```
{% block content %}
```

```
❶ <p><a href="{% url 'learning_logs:topic' topic.id %}">{{ topic }}</a></p>
```

```
<p>Add a new entry:</p>
```

```
❷ <form action="{% url 'learning_logs:new_entry' topic.id %}" method='post'>
```

```
  {% csrf_token %}
```

```
  {{ form.as_div }}
```

```
  <button name='submit'>Add entry</button>
```

```
</form>
```

```
{% endblock content %}
```

Mostramos el tema en la parte superior de la página ❶ , para que el usuario pueda ver a qué tema está añadiendo una entrada. El tema también actúa como enlace a la página principal de ese tema.

We show the topic at the top of the page ❶ , so the user can see which topic they're adding an entry to. The topic also acts as a link back to the main page for that topic.

El argumento `action` del formulario incluye el valor `topic.id` en la URL, para que la función de visualización pueda asociar la nueva entrada con el tema correcto ❷. Aparte de eso, esta plantilla es igual que `new_topic.html`.

The form's `action` argument includes the `topic.id` value in the URL, so the view function can associate the new entry with the correct topic ❷. Other than that, this template looks just like `new_topic.html`.

Enlace a la página nueva_entrada (Linking to the new_entry Page)

A continuación, tenemos que incluir un enlace a la página `new_entry` desde cada página de tema, en la plantilla de tema:

Next, we need to include a link to the `new_entry` page from each topic page, in the topic template:

topic.xhtml

topic.xhtml

```
{% extends "learning_logs/base.xhtml" %}
```

```
{% block content %}
```

```
<p>Topic: {{ topic }}</p>
```

```
<p>Entries:</p>
```

```
<p>
```

```
<a href="{% url 'learning_logs:new_entry' topic.id %}">Add new entry</a>
```

```
</p>
```

```
<ul>
```

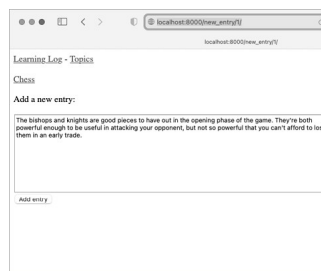
```
--snip--
```

```
</ul>
```

```
{% endblock content %}
```

Colocamos el enlace para añadir entradas justo antes de mostrar las entradas, porque añadir una nueva entrada será la acción más común en esta página. [La Figura 19-2](#) muestra la página `new_entry`. Ahora los usuarios pueden añadir nuevos temas y tantas entradas como deseen para cada tema. Prueba la página `new_entry` añadiendo algunas entradas a algunos de los temas que has creado.

We place the link to add entries just before showing the entries, because adding a new entry will be the most common action on this page. [Figure 19-2](#) shows the `new_entry` page. Now users can add new topics and as many entries as they want for each topic. Try out the `new_entry` page by adding a few entries to some of the topics you've created.



DESCRIPCIÓN DE LA IMAGEN (En español):

Registrar de Aprendizaje - Temas
Ajedrez Añadir una nueva entrada: Los alfiles y los caballos son buenas piezas para sacar en la fase de apertura del juego. Ambos son lo suficientemente poderosos para resultar útiles al atacar a tu oponente, pero no tan poderosos como para no poder permitirse perderlos en un intercambio temprano. Añadir entrada

Figura 19-2 : La página `new_entry`

Figure 19-2 : The `new_entry` page

[Editar entradas \(Editing Entries\)](#)

Ahora crearemos una página para que los usuarios puedan editar las entradas que han añadido.

Now we'll make a page so users can edit the entries they've added.

La URL `editar_entrada` (The `edit_entry` URL)

La URL de la página tiene que pasar el ID de la entrada que se va a editar. Aquí tienes `learning_logs/urls.py` :

The URL for the page needs to pass the ID of the entry to be edited. Here's `learning_logs/urls.py` :

urls.py

urls.py

```
--snip--
urlpatterns = [
    --snip--
    # Page for editing an entry.
    path('edit_entry/<int:entry_id>/', views.edit_entry, name='edit_entry'),
]
```

Este patrón de URL coincide con URL como `http://localhost:8000/edit_entry/id/` . Aquí el valor de `id` se asigna al parámetro `entry_id` . Django envía las solicitudes que coinciden con este formato a la función de vista `edit_entry()` .

This URL pattern matches URLs like `http://localhost:8000/edit_entry/id/` . Here the value of `id` is assigned to the parameter `entry_id` . Django sends requests that match this format to the view function `edit_entry()` .

La función de vista `edit_entry()` (The `edit_entry()` View Function)

Cuando la página `edit_entry` recibe una petición GET, la función `edit_entry()` devuelve un formulario para editar la entrada. Cuando la página recibe una petición POST con el texto de la entrada revisado, guarda el texto modificado en la base de datos:

When the `edit_entry` page receives a GET request, the `edit_entry()` function returns a form for editing the entry. When the page receives a POST request with revised entry text, it saves the modified text into the database:

vistas.py

views.py

```
from django.shortcuts import render, redirect

from .models import Topic, Entry
from .forms import TopicForm, EntryForm
--snip--

def edit_entry(request, entry_id):
    """Edit an existing entry."""
    ❶ entry = Entry.objects.get(id=entry_id)
    topic = entry.topic

    if request.method != 'POST':
        # Initial request; pre-fill form with the current entry.
        ❷ form = EntryForm(instance=entry)
    else:
        # POST data submitted; process data.
        ❸ form = EntryForm(instance=entry, data=request.POST)
        if form.is_valid():
            ❹ form.save()
            ❺ return redirect('learning_logs:topic', topic_id=topic.id)

    context = {'entry': entry, 'topic': topic, 'form': form}
    return render(request, 'learning_logs/edit_entry.xhtml', context)
```

Primero importamos el modelo `Entry`. A continuación, obtenemos el objeto entrada que el usuario desea editar ❶ y el tema asociado a esta entrada. En el bloque `if`, que se ejecuta para una solicitud GET, creamos una instancia de `EntryForm` con el argumento `instance=entry` ❷. Este argumento indica a Django que cree el formulario, relleno previamente con la información del objeto de entrada existente. El usuario verá sus datos existentes y podrá editarlos.

We first import the `Entry` model. We then get the entry object that the user wants to edit ❶ and the topic associated with this entry. In the `if` block, which runs for a GET request, we make an instance of `EntryForm` with the argument `instance=entry` ❷. This argument tells Django to create the form, prefilled with information from the existing entry object. The user will see their existing data and be able to edit that data.

Al procesar una solicitud POST, pasamos los argumentos `instance=entry` y `data=request.POST` ❸. Estos argumentos le dicen a Django que cree una instancia de formulario basada en la información asociada al objeto de entrada existente, actualizada con cualquier dato relevante de `request.POST`. A continuación, comprobamos si el formulario es válido; si lo es, llamamos a `save()` sin argumentos porque la entrada ya está asociada al tema correcto ❹. A continuación, redirigimos a la página `topic`, donde el usuario debería ver la versión actualizada de la entrada que ha editado ❺.

When processing a POST request, we pass both the `instance=entry` and the `data=request.POST` arguments ❸. These arguments tell Django to create a form instance based on the information associated with the existing entry object, updated with any relevant data from `request.POST`. We then check whether the form is valid; if it is, we call `save()` with no arguments because the entry is already associated with the correct topic ❹. We then redirect to the `topic` page, where the user should see the updated version of the entry they edited ❺.

Si mostramos un formulario inicial para editar la entrada o si el formulario enviado no es válido, creamos el diccionario `context` y renderizamos la página utilizando la plantilla `edit_entry.xhtml`.

If we're showing an initial form for editing the entry or if the submitted form is invalid, we create the `context` dictionary and render the page using the `edit_entry.xhtml` template.

La plantilla `edit_entry` (The `edit_entry` Template)

A continuación, creamos una plantilla `edit_entry.xhtml`, que es similar a `new_entry.xhtml`:

Next, we create an `edit_entry.xhtml` template, which is similar to `new_entry.xhtml`:

edit_entry.xhtml

edit_entry.xhtml

```
{% extends "learning_logs/base.xhtml" %}

{% block content %}

<p><a href="{% url 'learning_logs:topic' topic.id %}">{{ topic }}</a></p>

<p>Edit entry:</p>

❶ <form action="{% url 'learning_logs:edit_entry' entry.id %}" method='post'>
    {% csrf_token %}
    {{ form.as_div }}
❷ <button name="submit">Save changes</button>
</form>

{% endblock content %}
```

El argumento `action` devuelve el formulario a la función `edit_entry()` para que lo procese ❶. Incluimos el `entry.id` como argumento en la etiqueta `{% url %}`, para que la función de vista pueda modificar el objeto de entrada correcto. Etiquetamos el botón de envío como `Save changes` para recordar al usuario que está guardando ediciones, no creando una nueva entrada ❷.

The `action` argument sends the form back to the `edit_entry()` function for processing ❶. We include the `entry.id` as an argument in the `{% url %}` tag, so the view function can modify the correct entry object. We label the submit button as `Save changes` to remind the user they're saving edits, not creating a new entry ❷.

[Enlace a la página `edit\_entry` \(Linking to the `edit\_entry` Page\)](#)

Ahora tenemos que incluir un enlace a la página `edit_entry` para cada entrada de la página de temas:

Now we need to include a link to the `edit_entry` page for each entry on the topic page:

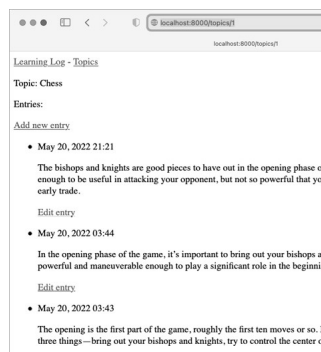
tema.xhtml

topic.xhtml

```
--snip--
{% for entry in entries %}
  <li>
    <p>{{ entry.date_added | date:'M d, Y H:i' }}</p>
    <p>{{ entry.text | linebreaks }}</p>
    <p>
      <a href="{% url 'learning_logs:edit_entry' entry.id %}">
        Edit entry</a></p>
    </li>
--snip--
```

Incluimos el enlace de edición después de mostrar la fecha y el texto de cada entrada. Utilizamos la etiqueta de plantilla `{% url %}` para determinar la URL del patrón de URL con nombre `edit_entry`, junto con el atributo ID de la entrada actual en el bucle (`entry.id`). El texto del enlace `Edit entry` aparece después de cada entrada de la página. [La Figura 19-3](#) muestra el aspecto de la página del tema con estos enlaces.

We include the edit link after each entry's date and text has been displayed. We use the `{% url %}` template tag to determine the URL for the named URL pattern `edit_entry`, along with the ID attribute of the current entry in the loop (`entry.id`). The link text `Edit entry` appears after each entry on the page. [Figure 19-3](#) shows what the topic page looks like with these links.



DESCRIPCIÓN DE LA
IMAGEN (En español):

Registro de
Aprendizaje - Temas
Tema: Ajedrez
Entradas: Añadir nueva
entrada 20 de mayo de
2022 21:21 Los alfiles
y los caballos son
buenas piezas para
abrir el juego. Ambos
son lo suficientemente

poderosos como para ser útiles al atacar a tu oponente, pero no tan poderosos como para que te preocupe perderlos temprano en una partida. Editar entrada 20 de mayo de 2022 03:44 En la apertura del juego, es importante llevar a cabo tanto a tus alfiles como a tus caballos. Estas piezas son poderosas y pueden maniobrar con éxito un papel estratégico en las primeras etapas de una partida. Editar entrada 20 de mayo de 2022 03:43 La apertura es la primera parte del juego, aproximadamente los primeros diez movimientos o así. En la apertura, es una buena idea llevar tanto a tus alfiles como a tus caballos, tratando de controlar el centro del tablero y establecer un juego claro.

Figura 19-3 : Cada entrada tiene ahora un enlace para editarla.

Figure 19-3 : Each entry now has a link for editing that entry.

El Registro de Aprendizaje tiene ahora la mayor parte de la funcionalidad que necesita. Los usuarios pueden añadir temas y entradas, y pueden leer cualquier conjunto de entradas que deseen. En la próxima sección, implementaremos un sistema de registro de usuarios para que cualquiera pueda crear una cuenta en Learning Log y crear su propio conjunto de temas y entradas.

Learning Log now has most of the functionality it needs. Users can add topics and entries, and they can read through any set of entries they want. In the next section, we'll implement a user registration system so anyone can make an account with Learning Log and create their own set of topics and entries.

Pruébalo tú mismo (*Try It Yourself*)

19-1. Blog: Inicia un nuevo proyecto Django llamado *Blog* . Crea una aplicación llamada *blogs* , con un modelo que represente un blog en general, y un modelo que represente una entrada individual del blog. Asigna a cada modelo un conjunto adecuado de campos. Crea un superusuario para el proyecto y utiliza el sitio de administración para crear un blog y un par de entradas cortas. Crea una página de inicio que muestre todas las entradas en el orden adecuado.

19-1. Blog: Start a new Django project called *Blog* . Create an app called *blogs* , with one model that represents an overall blog, and one model that represents an individual blog post. Give each model an appropriate set of fields. Create a superuser for the project, and use the admin site to make a blog and a couple of short posts. Make a home page that shows all posts in an appropriate order.

Crea páginas para crear un blog, para crear nuevas entradas y para editar las entradas existentes. Utiliza tus páginas para asegurarte de que funcionan.

Create pages for making a blog, for making new posts, and for editing existing posts. Use your pages to make sure they work.

Configurar cuentas de usuario (*Setting Up User Accounts*)

En esta sección, configuraremos un sistema de registro y autorización de usuarios para que la gente pueda registrar una cuenta, entrar y salir. Crearemos una nueva app que contenga toda la funcionalidad relacionada con el trabajo con usuarios. Utilizaremos el sistema de autenticación de usuarios por defecto incluido en Django para realizar la mayor parte del trabajo posible. También modificaremos ligeramente el modelo `Topic` para que cada tema pertenezca a un usuario determinado.

In this section, we'll set up a user registration and authorization system so people can register an account, log in, and log out. We'll create a new app to contain all the functionality related to working with users. We'll use the default user authentication system included with Django to do as much of the work as possible. We'll also modify the `Topic` model slightly so every topic belongs to a certain user.

La App de cuentas (*The accounts App*)

Empezaremos creando una nueva app llamada `accounts` , utilizando el comando `startapp` :

We'll start by creating a new app called `accounts` , using the `startapp` command:

```
(ll_env)learning_log$ python manage.py startapp accounts
(ll_env)learning_log$ ls
❶ accounts db.sqlite3 learning_logs ll_env ll_project manage.py
(ll_env)learning_log$ ls accounts
❷ __init__.py admin.py apps.py migrations models.py tests.py views.py
```

El sistema de autenticación por defecto se basa en el concepto de cuentas de usuario, por lo que utilizar el nombre `accounts` facilita la integración con el sistema por defecto. El comando `startapp` que se muestra aquí crea un nuevo directorio llamado *accounts* ❶ con una estructura idéntica a la app `learning_logs` ❷ .

The default authentication system is built around the concept of user accounts, so using the name `accounts` makes integration with the default system easier. The `startapp` command shown here makes a new directory called *accounts* ❶ with a structure identical to the `learning_logs` app ❷ .

Añadir cuentas a *settings.py* (*Adding accounts to settings.py*)

Tenemos que añadir nuestra nueva app a `INSTALLED_APPS` en *settings.py* , de la siguiente manera:

We need to add our new app to `INSTALLED_APPS` in `settings.py`, like so:

settings.py

settings.py

```
--snip--
INSTALLED_APPS = [
    # My apps
    'learning_logs',
    'accounts',

    # Default django apps.
    --snip--
]
```

Ahora Django incluirá la app `accounts` en el proyecto global.

Now Django will include the `accounts` app in the overall project.

Incluir las URL de las cuentas (Including the URLs from accounts)

A continuación, tenemos que modificar la raíz `urls.py` para que incluya las URL que escribiremos para la app `accounts` :

Next, we need to modify the root `urls.py` so it includes the URLs we'll write for the `accounts` app:

ll_project/urls.py

ll_project/urls.py

```
from django.contrib import admin
from django.urls import path, include
```

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('accounts/', include('accounts.urls')),
    path("", include('learning_logs.urls')),
]
```

Añadimos una línea para incluir el archivo *urls.py* de *accounts* . Esta línea coincidirá con cualquier URL que empiece por la palabra *accounts* , como *http://localhost:8000/accounts/login/* .

We add a line to include the file *urls.py* from *accounts* . This line will match any URL that starts with the word *accounts* , such as *http://localhost:8000/accounts/login/* .

La página de inicio de sesión (*The Login Page*)

Primero implementaremos una página de inicio de sesión. Utilizaremos la vista *login* predeterminada que proporciona Django, por lo que el patrón de URL para esta aplicación es un poco diferente. Crea un nuevo archivo *urls.py* en el directorio *ll_project/accounts/* y añádele lo siguiente:

We'll first implement a login page. We'll use the default *login* view Django provides, so the URL pattern for this app looks a little different. Make a new *urls.py* file in the directory *ll_project/accounts/* and add the following to it:

cuentas/urls.py

accounts/urls.py

```
"""Defines URL patterns for accounts."""
```

```
from django.urls import path, include
```

```
app_name = 'accounts'
```

```
urlpatterns = [  
    # Include default auth urls.  
    path("", include('django.contrib.auth.urls')),  
]
```

Importamos la función `path` y, a continuación, importamos la función `include` para poder incluir algunas URL de autenticación por defecto que Django ha definido. Estas URL predeterminadas incluyen patrones de URL con nombre, como `'login'` y `'logout'` . Establecemos la variable `app_name` en `'accounts'` para que Django pueda distinguir estas URL de las que pertenecen a otras aplicaciones. Incluso las URL predeterminadas proporcionadas por Django, cuando se incluyan en el archivo `urls.py` de la app `accounts` , serán accesibles a través del espacio de nombres `accounts` .

We import the `path` function, and then import the `include` function so we can include some default authentication URLs that Django has defined. These default URLs include named URL patterns, such as `'login'` and `'logout'` . We set the variable `app_name` to `'accounts'` so Django can distinguish these URLs from URLs belonging to other apps. Even default URLs provided by Django, when included in the `accounts` app's `urls.py` file, will be accessible through the `accounts` namespace.

El patrón de la página de acceso coincide con la URL `http://localhost:8000/accounts/login/` . Cuando Django lee esta URL, la palabra `accounts` indica a Django que busque en `accounts/urls.py` , y `login` le indica que envíe las peticiones a la vista por defecto de Django `login` .

The login page's pattern matches the URL `http://localhost:8000/accounts/login/` . When Django reads this URL, the word `accounts` tells Django to look in `accounts/urls.py` , and `login` tells it to send requests to Django's default `login` view.

La plantilla de inicio de sesión (The login Template)

Cuando el usuario solicite la página de inicio de sesión, Django utilizará una función de vista por defecto, pero aún tenemos que proporcionar una plantilla para la página. Las vistas de autenticación por defecto buscan plantillas dentro de una carpeta llamada `registration` , así que tendremos que crear esa carpeta. Dentro del directorio `ll_project/accounts/` , crea un directorio llamado `templates` ; dentro de éste, crea otro directorio llamado `registration` . Aquí está la plantilla `login.xhtml` , que debería guardarse en `ll_project/accounts/templates/registration` :

When the user requests the login page, Django will use a default view function, but we still need to provide a template for the page. The default authentication views look for templates inside a folder called *registration*, so we'll need to make that folder. Inside the *ll_project/accounts/* directory, make a directory called *templates*; inside that, make another directory called *registration*. Here's the *login.html* template, which should be saved in *ll_project/accounts/templates/registration*:

login.html

login.html

```
{% extends 'learning_logs/base.html' %}

{% block content %}

❶  {% if form.errors %}
    <p>Your username and password didn't match. Please try again.</p>
    {% endif %}

❷  <form action="{% url 'accounts:login' %}" method='post'>
    {% csrf_token %}
❸  {{ form.as_div }}

❹  <button name="submit">Log in</button>
    </form>

{% endblock content %}
```

Esta plantilla amplía *base.html* para garantizar que la página de inicio de sesión tenga el mismo aspecto que el resto del sitio. Ten en cuenta que una plantilla de una aplicación puede heredar de una plantilla de otra aplicación.

This template extends *base.html* to ensure that the login page will have the same look and feel as the rest of the site. Note that a template in one app can inherit from a template in another app.

Si se establece el atributo `errors` del formulario, mostramos un mensaje de error ❶ , informando de que la combinación de nombre de usuario y contraseña no coincide con nada almacenado en la base de datos.

If the form's `errors` attribute is set, we display an error message ❶ , reporting that the username and password combination doesn't match anything stored in the database.

Queremos que la vista de inicio de sesión procese el formulario, así que establecemos el argumento `action` como la URL de la página de inicio de sesión ❷ . La vista de inicio de sesión envía un objeto `form` a la plantilla, y nos corresponde a nosotros mostrar el formulario ❸ y añadir un botón de envío ❹ .

We want the login view to process the form, so we set the `action` argument as the URL of the login page ❷ . The login view sends a `form` object to the template, and it's up to us to display the form ❸ and add a submit button ❹ .

La configuración de `LOGIN_REDIRECT_URL` (The `LOGIN_REDIRECT_URL` Setting)

Una vez que un usuario se registra correctamente, Django necesita saber dónde enviarlo. Esto lo controlamos en el archivo de configuración.

Once a user logs in successfully, Django needs to know where to send that user. We control this in the settings file.

Añade el siguiente código al final de *settings.py* :

Add the following code to the end of *settings.py* :

settings.py

settings.py

--snip--

My settings.

LOGIN_REDIRECT_URL = 'learning_logs:index'

Con todos los ajustes por defecto en *settings.py*, es útil marcar la sección en la que vamos a añadir nuevos ajustes. La primera configuración nueva que añadiremos es `LOGIN_REDIRECT_URL`, que indica a Django a qué URL debe redirigir después de un intento de inicio de sesión con éxito.

With all the default settings in *settings.py*, it's helpful to mark off the section where we're adding new settings. The first new setting we'll add is `LOGIN_REDIRECT_URL`, which tells Django which URL to redirect to after a successful login attempt.

Enlace a la página de inicio de sesión (Linking to the Login Page)

Vamos a añadir el enlace de inicio de sesión a *base.html* para que aparezca en todas las páginas. No queremos que el enlace aparezca cuando el usuario ya haya iniciado sesión, así que lo anidaremos dentro de una etiqueta `{% if %}`:

Let's add the login link to *base.html* so it appears on every page. We don't want the link to display when the user is already logged in, so we nest it inside an `{% if %}` tag:

base.html

base.html

<p>

Learning Log -

Topics -

❶ {% if user.is_authenticated %}

❷ Hello, {{ user.username }}.

{% else %}

❸ Log in

{% endif %}

</p>

{% block content %}{% endblock content %}

En el sistema de autenticación de Django, cada plantilla dispone de un objeto `user` que siempre tiene definido el atributo `is_authenticated` : el atributo es `True` si el usuario está conectado y `False` si no lo está. Este atributo te permite mostrar un mensaje a los usuarios autenticados y otro a los no autenticados.

In Django's authentication system, every template has a `user` object available that always has an `is_authenticated` attribute set: the attribute is `True` if the user is logged in and `False` if they aren't. This attribute allows you to display one message to authenticated users and another to unauthenticated users.

Aquí mostramos un saludo a los usuarios conectados ❶ . Los usuarios autenticados tienen un atributo adicional `username` , que utilizamos para personalizar el saludo y recordar al usuario que ha iniciado sesión ❷ . Para los usuarios que no se han autenticado, mostramos un enlace a la página de inicio de sesión ❸ .

Here we display a greeting to users currently logged in ❶ . Authenticated users have an additional `username` attribute set, which we use to personalize the greeting and remind the user they're logged in ❷ . For users who haven't been authenticated, we display a link to the login page ❸ .

Uso de la página de inicio de sesión (Using the Login Page)

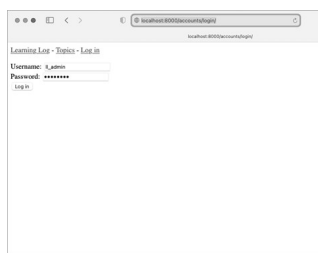
Ya hemos configurado una cuenta de usuario, así que vamos a iniciar sesión para ver si la página funciona. Ve a <http://localhost:8000/admin/> . Si todavía estás conectado como administrador, busca un enlace de **cierre de sesión** en la cabecera y haz clic en él.

We've already set up a user account, so let's log in to see if the page works. Go to <http://localhost:8000/admin/> . If you're still logged in as an admin, look for a **logout** link in the header and click it.

Cuando hayas cerrado la sesión, ve a <http://localhost:8000/accounts/login/> . Deberías ver una página de inicio de sesión similar a la que se muestra en [la Figura 19-4](#) . Introduce el nombre de usuario y la contraseña que configuraste anteriormente, y deberías volver a la

página de inicio. La cabecera de la página de inicio debería mostrar un saludo personalizado con tu nombre de usuario.

When you're logged out, go to `http://localhost:8000/accounts/login/`. You should see a login page similar to the one shown in [Figure 19-4](#). Enter the username and password you set up earlier, and you should be brought back to the home page. The header on the home page should display a greeting personalized with your username.



DESCRIPCIÓN DE LA
IMAGEN (En español):

Nombre de usuario:
admin Contraseña: ****
Iniciar sesión Registro
de Aprendizaje - Temas
- Iniciar sesión

Figura 19-4 : Página de inicio de sesión

Figure 19-4 : The login page

Cerrar sesión (Logging Out)

Ahora tenemos que proporcionar a los usuarios una forma de cerrar la sesión. Las solicitudes de cierre de sesión deben enviarse como solicitudes POST, así que añadiremos un pequeño formulario de cierre de sesión a `base.html`. Cuando los usuarios pulsen el botón de cierre de sesión, irán a una página que confirme que han cerrado la sesión.

Now we need to provide a way for users to log out. Logout requests should be submitted as POST requests, so we'll add a small logout form to `base.html`. When users click the logout button, they'll go to a page confirming that they've been logged out.

Añadir un formulario de cierre de sesión a `base.html` (Adding a Logout Form to `base.html`)

Añadiremos el formulario de cierre de sesión a `base.html` para que esté disponible en todas las páginas. Lo incluiremos en otro bloque de `if`, para que sólo puedan verlo los usuarios que ya hayan iniciado sesión:

We'll add the form for logging out to `base.html` so it's available on every page. We'll include it in another `if` block, so only users who are already logged in can see it:

base.xhtml

base.xhtml

```
--snip--
{% block content %}{% endblock content %}

{% if user.is_authenticated %}
❶ <hr />
❷ <form action="{% url 'accounts:logout' %}" method='post'>
    {% csrf_token %}
    <button name='submit'>Log out</button>
</form>
{% endif %}
```

El patrón de URL por defecto para cerrar la sesión es 'accounts/logout/'. Sin embargo, la solicitud debe enviarse como una solicitud POST; de lo contrario, los atacantes pueden forzar fácilmente las solicitudes de cierre de sesión. Para que la solicitud de cierre de sesión utilice POST, definimos un formulario sencillo.

The default URL pattern for logging out is 'accounts/logout/'. However, the request has to be sent as a POST request; otherwise, attackers can easily force logout requests. To make the logout request use POST, we define a simple form.

Colocamos el formulario en la parte inferior de la página, debajo de un elemento de regla horizontal (<hr />) ❶ . Esta es una forma sencilla de mantener siempre el botón de cierre de sesión en una posición coherente debajo de cualquier otro contenido de la página. El propio formulario tiene la URL de cierre de sesión como argumento `action` , y 'post' como método de solicitud ❷ . Todo formulario en Django necesita incluir el `{% csrf_token %}` , incluso un formulario sencillo como éste. Este formulario está vacío excepto por el botón de envío.

We place the form at the bottom of the page, below a horizontal rule element (<hr />) ❶ . This is an easy way to always keep the logout button in a consistent position below any other content on the page. The form itself has the logout URL as its `action` argument,

and 'post' as the request method ❷ . Every form in Django needs to include the `{% csrf_token %}` , even a simple form like this one. This form is empty except for the submit button.

El ajuste LOGOUT_REDIRECT_URL (The LOGOUT_REDIRECT_URL Setting)

Cuando el usuario hace clic en el botón de cierre de sesión, Django necesita saber a dónde enviarlo. Controlamos este comportamiento en *settings.py* :

When the user clicks the logout button, Django needs to know where to send them. We control this behavior in *settings.py* :

settings.py

settings.py

```
--snip--
# My settings.
LOGIN_REDIRECT_URL = 'learning_logs:index'
LOGOUT_REDIRECT_URL = 'learning_logs:index'
```

La configuración de `LOGOUT_REDIRECT_URL` que se muestra aquí indica a Django que redirija a los usuarios que han cerrado la sesión a la página de inicio. Esta es una forma sencilla de confirmar que han cerrado la sesión, ya que no deberían ver su nombre de usuario después de cerrar la sesión.

The `LOGOUT_REDIRECT_URL` setting shown here tells Django to redirect logged-out users back to the home page. This is a simple way to confirm that they were logged out, because they should no longer see their username after logging out.

La página de registro (The Registration Page)

A continuación, crearemos una página para que los nuevos usuarios puedan registrarse. Utilizaremos la página predeterminada de Django `UserCreationForm` , pero escribiremos nuestra propia función de vista y plantilla.

Next, we'll build a page so new users can register. We'll use Django's default `UserCreationForm` , but write our own view function and template.

La URL de registro (The register URL)

El siguiente código proporciona el patrón de URL para la página de registro, que debe colocarse en *accounts/urls.py* :

The following code provides the URL pattern for the registration page, which should be placed in *accounts/urls.py* :

cuentas/urls.py

accounts/urls.py

```
"""Defines URL patterns for accounts."""

from django.urls import path, include

from . import views

app_name = accounts
urlpatterns = [
    # Include default auth urls.
    path("", include('django.contrib.auth.urls')),
    # Registration page.
    path('register/', views.register, name='register'),
]
```

Importamos el módulo `views` de `accounts` , que necesitamos porque estamos escribiendo nuestra propia vista para la página de registro. El patrón de la página de registro coincide con la URL `http://localhost:8000/accounts/register/` y envía peticiones a la función `register()` que vamos a escribir.

We import the `views` module from `accounts`, which we need because we're writing our own view for the registration page. The pattern for the registration page matches the URL `http://localhost:8000/accounts/register/` and sends requests to the `register()` function we're about to write.

La función de la vista `register()` (The `register()` View Function)

La función de vista `register()` tiene que mostrar un formulario de registro en blanco cuando se solicite la página de registro por primera vez, y luego procesar los formularios de registro completados cuando se envíen. Cuando un registro se realiza correctamente, la función también debe iniciar la sesión del nuevo usuario. Añade el siguiente código a `accounts/views.py`:

The `register()` view function needs to display a blank registration form when the registration page is first requested, and then process completed registration forms when they're submitted. When a registration is successful, the function also needs to log the new user in. Add the following code to `accounts/views.py`:

`cuentas/vistas.py`

`accounts/views.py`

```
from django.shortcuts import render, redirect
from django.contrib.auth import login
from django.contrib.auth.forms import UserCreationForm
```

```
def register(request):
    """Register a new user."""
    if request.method != 'POST':
        # Display blank registration form.
        ❶ form = UserCreationForm()
    else:
        # Process completed form.
        ❷ form = UserCreationForm(data=request.POST)

        ❸ if form.is_valid():
            ❹ new_user = form.save()
```

```

        # Log the user in and then redirect to home page.
        5 login(request, new_user)
        6 return redirect('learning_logs:index')

# Display a blank or invalid form.
context = {'form': form}
return render(request, 'registration/register.xhtml', context)

```

Importamos las funciones `render()` y `redirect()`, y luego importamos la función `login()` para iniciar la sesión del usuario si su información de registro es correcta. También importamos la función por defecto `UserCreationForm`. En la función `register()`, comprobamos si estamos respondiendo a una solicitud POST. Si no es así, creamos una instancia de `UserCreationForm` sin datos iniciales ❶.

We import the `render()` and `redirect()` functions, and then we import the `login()` function to log the user in if their registration information is correct. We also import the default `UserCreationForm`. In the `register()` function, we check whether we're responding to a POST request. If we're not, we make an instance of `UserCreationForm` with no initial data ❶.

Si estamos respondiendo a una solicitud POST, creamos una instancia de `UserCreationForm` basada en los datos enviados ❷. Comprobamos que los datos son válidos ❸ -en este caso, que el nombre de usuario tiene los caracteres apropiados, las contraseñas coinciden y el usuario no está intentando hacer nada malicioso en su envío.

If we're responding to a POST request, we make an instance of `UserCreationForm` based on the submitted data ❷. We check that the data is valid ❸ —in this case, that the username has the appropriate characters, the passwords match, and the user isn't trying to do anything malicious in their submission.

Si los datos enviados son válidos, llamamos al método `save()` del formulario para guardar el nombre de usuario y el hash de la contraseña en la base de datos ❹. El método `save()` devuelve el objeto usuario recién creado, que asignamos a `new_user`. Una vez guardada la información del usuario, iniciamos su sesión llamando a la función `login()` con los objetos `request` y `new_user` ❺, que crea una sesión válida para el nuevo usuario. Por último, redirigimos al usuario a la página de inicio ❻, donde un saludo personalizado en la cabecera le informa de que su registro se ha realizado correctamente.

If the submitted data is valid, we call the form's `save()` method to save the username and the hash of the password to the database ④. The `save()` method returns the newly created user object, which we assign to `new_user`. When the user's information is saved, we log them in by calling the `login()` function with the `request` and `new_user` objects ⑤, which creates a valid session for the new user. Finally, we redirect the user to the home page ⑥, where a personalized greeting in the header tells them their registration was successful.

Al final de la función, mostramos la página, que será un formulario en blanco o un formulario enviado que no es válido.

At the end of the function, we render the page, which will be either a blank form or a submitted form that's invalid.

La plantilla de registro (The register Template)

Ahora crea una plantilla para la página de registro, que será similar a la página de inicio de sesión. Asegúrate de guardarla en el mismo directorio que *login.xhtml*:

Now create a template for the registration page, which will be similar to the login page. Be sure to save it in the same directory as *login.xhtml*:

registro.xhtml

register.xhtml

```
{% extends "learning_logs/base.xhtml" %}

{% block content %}

<form action="{% url 'accounts:register' %}" method='post'>
  {% csrf_token %}
  {{ form.as_div }}

  <button name="submit">Register</button>
```

```
</form>
```

```
{% endblock content %}
```

Esto debería parecerse a las otras plantillas basadas en formularios que hemos estado escribiendo. Volvemos a utilizar el método `as_div` para que Django muestre adecuadamente todos los campos del formulario, incluidos los mensajes de error si no se rellena correctamente.

This should look like the other form-based templates we've been writing. We use the `as_div` method again so Django will display all the fields in the form appropriately, including any error messages if the form isn't filled out correctly.

Enlace a la página de registro (Linking to the Registration Page)

A continuación, añadiremos código para mostrar el enlace a la página de registro a cualquier usuario que no haya iniciado sesión:

Next, we'll add code to show the registration page link to any user who isn't currently logged in:

base.xhtml

base.xhtml

```
--snip--
{% if user.is_authenticated %}
    Hello, {{ user.username }}.
{% else %}
    <a href="{% url 'accounts:register' %}">Register</a> -
    <a href="{% url 'accounts:login' %}">Log in</a>
{% endif %}
--snip--
```

Ahora, los usuarios que han iniciado sesión ven un saludo personalizado y un botón de cierre de sesión. Los usuarios que no han iniciado sesión ven un enlace de registro y un enlace de inicio de sesión. Prueba la página de registro creando varias cuentas de usuario

con diferentes nombres de usuario.

Now users who are logged in see a personalized greeting and a logout button. Users who aren't logged in see a registration link and a login link. Try out the registration page by making several user accounts with different usernames.

En la siguiente sección, restringiremos algunas de las páginas para que sólo estén disponibles para los usuarios registrados, y nos aseguraremos de que cada tema pertenezca a un usuario concreto.

In the next section, we'll restrict some of the pages so they're available only to registered users, and we'll make sure every topic belongs to a specific user.

Nota (Note)

El sistema de registro que hemos configurado permite a cualquiera crear cualquier número de cuentas para Learning Log. Algunos sistemas requieren que los usuarios confirmen su identidad mediante el envío de un correo electrónico de confirmación al que los usuarios deben responder. De este modo, el sistema genera menos cuentas de spam que el sistema sencillo que utilizamos aquí. Sin embargo, cuando estás aprendiendo a crear aplicaciones, es perfectamente apropiado practicar con un sistema sencillo de registro de usuarios como el que estamos utilizando.

The registration system we've set up allows anyone to make any number of accounts for Learning Log. Some systems require users to confirm their identity by sending a confirmation email that users must reply to. By doing so, the system generates fewer spam accounts than the simple system we're using here. However, when you're learning to build apps, it's perfectly appropriate to practice with a simple user registration system like the one we're using.

Pruébalo tú mismo (Try It Yourself)

19-2. Cuentas de Blog: Añade un sistema de autenticación y registro de usuarios al proyecto Blog que iniciaste en el Ejercicio 19-1 (página 415). Asegúrate de que los usuarios registrados ven su nombre de usuario en algún lugar de la pantalla y que los usuarios no registrados ven un enlace a la página de registro.

19-2. Blog Accounts: Add a user authentication and registration system to the Blog project you started in Exercise 19-1 (page 415). Make sure logged-in users see their username somewhere on the screen and unregistered users see a link to the registration page.

Permitir que los usuarios sean propietarios de sus datos (Allowing Users to Own Their Data)

Los usuarios deben poder introducir datos privados en sus registros de aprendizaje, así que crearemos un sistema para averiguar qué datos pertenecen a cada usuario. Luego restringiremos el acceso a determinadas páginas para que los usuarios sólo puedan trabajar con sus propios datos.

Users should be able to enter private data in their learning logs, so we'll create a system to figure out which data belongs to which user. Then we'll restrict access to certain pages so users can only work with their own data.

Modificaremos el modelo `Topic` para que cada tema pertenezca a un usuario concreto. Esto también se ocupará de las entradas, porque cada entrada pertenece a un tema concreto. Empezaremos restringiendo el acceso a determinadas páginas.

We'll modify the `Topic` model so every topic belongs to a specific user. This will also take care of entries, because every entry belongs to a specific topic. We'll start by restricting access to certain pages.

Restringir el acceso con `@login_required` (Restricting Access with `@login_required`)

Django facilita la restricción de acceso a determinadas páginas mediante el decorador `@login_required` . Recuerda del Capítulo 11 que un *decorator* es una directiva que se

coloca justo antes de la definición de una función y que modifica el comportamiento de ésta. Veamos un ejemplo.

Django makes it easy to restrict access to certain pages through the `@login_required` decorator. Recall from Chapter 11 that a *decorator* is a directive placed just before a function definition, which modifies how the function behaves. Let's look at an example.

Restringir el acceso a la página Temas (Restricting Access to the Topics Page)

Cada tema será propiedad de un usuario, por lo que sólo los usuarios registrados podrán solicitar la página de temas. Añade el siguiente código a *learning_logs/views.py* :

Each topic will be owned by a user, so only registered users can request the topics page. Add the following code to *learning_logs/views.py* :

learning_logs/views.py

learning_logs/views.py

```
from django.shortcuts import render, redirect
from django.contrib.auth.decorators import login_required
```

```
from .models import Topic, Entry
--snip--
```

```
@login_required
def topics(request):
    """Show all topics."""
    --snip--
```

Primero importamos la función `login_required()` . Aplicamos `login_required()` como decorador a la función de vista `topics()` anteponiendo a `login_required` el símbolo `@` . Como resultado, Python sabe que debe ejecutar el código de `login_required()` antes que el código de `topics()` .

We first import the `login_required()` function. We apply `login_required()` as a decorator to the `topics()` view function by prepending `login_required` with the `@` symbol. As a result, Python knows to run the code in `login_required()` before the code in `topics()` .

El código en `login_required()` comprueba si un usuario está conectado, y Django ejecuta el código en `topics()` sólo si lo está. Si el usuario no ha iniciado sesión, se le redirige a la página de inicio de sesión.

The code in `login_required()` checks whether a user is logged in, and Django runs the code in `topics()` only if they are. If the user isn't logged in, they're redirected to the login page.

Para que esta redirección funcione, tenemos que modificar *settings.py* para que Django sepa dónde encontrar la página de inicio de sesión. Añade lo siguiente al final de *settings.py* :

To make this redirect work, we need to modify *settings.py* so Django knows where to find the login page. Add the following at the end of *settings.py* :

settings.py

settings.py

```
--snip--
# My settings.
LOGIN_REDIRECT_URL = 'learning_logs:index'
LOGOUT_REDIRECT_URL = 'learning_logs:index'
LOGIN_URL = 'accounts:login'
```

Ahora, cuando un usuario no autenticado solicite una página protegida por el decorador `@login_required` , Django enviará al usuario a la URL definida por `LOGIN_URL` en *settings.py* .

Now when an unauthenticated user requests a page protected by the `@login_required` decorator, Django will send the user to the URL defined by `LOGIN_URL` in `settings.py`.

Puedes probar esta configuración cerrando la sesión de cualquier cuenta de usuario y accediendo a la página de inicio. Haz clic en el enlace **Temas**, que debería redirigirte a la página de inicio de sesión. A continuación, inicia sesión en cualquiera de tus cuentas y, desde la página de inicio, vuelve a hacer clic en el enlace **Temas**. Deberías poder acceder a la página de temas.

You can test this setting by logging out of any user accounts and going to the home page. Click the **Topics** link, which should redirect you to the login page. Then log in to any of your accounts, and from the home page, click the **Topics** link again. You should be able to access the topics page.

Restringir el acceso a través del Registro de Aprendizaje (Restricting Access Throughout Learning Log)

Django facilita la restricción de acceso a las páginas, pero tienes que decidir qué páginas proteger. Lo mejor es pensar primero qué páginas deben estar libres de restricciones, y luego restringir todas las demás páginas del proyecto. Puedes corregir fácilmente el acceso excesivamente restringido, y es menos peligroso que dejar páginas sensibles sin restringir.

Django makes it easy to restrict access to pages, but you have to decide which pages to protect. It's best to think about which pages need to be unrestricted first, and then restrict all the other pages in the project. You can easily correct over-restricted access, and it's less dangerous than leaving sensitive pages unrestricted.

En Learning Log, mantendremos sin restricciones la página de inicio y la página de registro. Restringiremos el acceso a todas las demás páginas.

In Learning Log, we'll keep the home page and the registration page unrestricted. We'll restrict access to every other page.

Aquí tienes `learning_logs/views.py` con los decoradores `@login_required` aplicados a todas las vistas excepto a `index()` :

Here's *learning_logs/views.py* with `@login_required` decorators applied to every view except `index()` :

learning_logs/views.py

learning_logs/views.py

```
--snip--
@login_required
def topics(request):
    --snip--

@login_required
def topic(request, topic_id):
    --snip--

@login_required
def new_topic(request):
    --snip--

@login_required
def new_entry(request, topic_id):
    --snip--

@login_required
def edit_entry(request, entry_id):
    --snip--
```

Intenta acceder a cada una de estas páginas mientras estás desconectado; deberías ser redirigido a la página de inicio de sesión. Tampoco podrás hacer clic en enlaces a páginas como `new_topic` . Pero si introduces la URL `http://localhost:8000/new_topic/` , serás redirigido a la página de inicio de sesión. Deberías restringir el acceso a cualquier URL que sea de acceso público y esté relacionada con datos privados del usuario.

Try accessing each of these pages while logged out; you should be redirected back to the login page. You'll also be unable to click links to pages such as `new_topic` . But if you enter

the URL `http://localhost:8000/new_topic/`, you'll be redirected to the login page. You should restrict access to any URL that's publicly accessible and relates to private user data.

Conexión de datos a determinados usuarios (*Connecting Data to Certain Users*)

A continuación, tenemos que conectar los datos al usuario que los envió. Sólo tenemos que conectar los datos más altos de la jerarquía a un usuario, y los datos de nivel inferior le seguirán. En Learning Log, los temas son el nivel más alto de datos de la aplicación, y todas las entradas están conectadas a un tema. Mientras cada tema pertenezca a un usuario concreto, podremos rastrear la propiedad de cada entrada en la base de datos.

Next, we need to connect the data to the user who submitted it. We only need to connect the data highest in the hierarchy to a user, and the lower-level data will follow. In Learning Log, topics are the highest level of data in the app, and all entries are connected to a topic. As long as each topic belongs to a specific user, we can trace the ownership of each entry in the database.

Modificaremos el modelo `Topic` añadiendo una relación de clave foránea a un usuario. A continuación, tendremos que migrar la base de datos. Por último, modificaremos algunas de las vistas para que sólo muestren los datos asociados al usuario conectado en ese momento.

We'll modify the `Topic` model by adding a foreign key relationship to a user. We'll then have to migrate the database. Finally, we'll modify some of the views so they only show the data associated with the currently logged-in user.

Modificar el modelo de temas (*Modifying the Topic Model*)

La modificación de `models.py` son sólo dos líneas:

The modification to `models.py` is just two lines:

modelos.py

models.py

```
from django.db import models
from django.contrib.auth.models import User

class Topic(models.Model):
    """A topic the user is learning about."""
    Text = models.CharField(max_length=200)
    date_added = models.DateTimeField(auto_now_add=True)
    owner = models.ForeignKey(User, on_delete=models.CASCADE)

    def __str__(self):
        """Return a string representing the topic."""
        Return self.text

class Entry(models.Model):
    --snip--
```

Importamos el modelo `User` de `django.contrib.auth`. A continuación, añadimos un campo `owner` a `Topic`, que establece una relación de clave foránea con el modelo `User`. Si se elimina un usuario, se eliminarán también todos los temas asociados a ese usuario.

We import the `User` model from `django.contrib.auth`. Then we add an `owner` field to `Topic`, which establishes a foreign key relationship to the `User` model. If a user is deleted, all the topics associated with that user will be deleted as well.

Identificar a los usuarios existentes (Identifying Existing Users)

Cuando migremos la base de datos, Django modificará la base de datos para que pueda almacenar una conexión entre cada tema y un usuario. Para realizar la migración, Django necesita saber qué usuario asociar a cada tema existente. Lo más sencillo es empezar asignando todos los temas existentes a un usuario, por ejemplo, el superusuario. Pero antes, necesitamos conocer el ID de ese usuario.

When we migrate the database, Django will modify the database so it can store a connection between each topic and a user. To make the migration, Django needs to know which user to associate with each existing topic. The simplest approach is to start by assigning all existing topics to one user—for example, the superuser. But first, we need to know that user's ID.

Veamos los ID de todos los usuarios creados hasta ahora. Inicia una sesión shell de Django y emite los siguientes comandos:

Let's look at the IDs of all users created so far. Start a Django shell session and issue the following commands:

```
(ll_env)learning_log$ python manage.py shell
❶ >>> from django.contrib.auth.models import User
❷ >>> User.objects.all()
<QuerySet [<User: ll_admin>, <User: eric>, <User: willie>]>
❸ >>> for user in User.objects.all():
...     print(user.username, user.id)
...
ll_admin 1
eric 2
willie 3
>>>
```

Primero importamos el modelo `User` a la sesión shell ❶ . A continuación, miramos todos los usuarios que se han creado hasta el momento ❷ . La salida muestra tres usuarios para mi versión del proyecto: `ll_admin` `eric` y `willie` .

We first import the `User` model into the shell session ❶ . We then look at all the users that have been created so far ❷ . The output shows three users for my version of the project: `ll_admin` , `eric` , and `willie` .

A continuación, recorreremos la lista de usuarios e imprimimos el nombre de usuario y el ID de cada usuario ❸ . Cuando Django pregunte a qué usuario asociar los temas existentes, utilizaremos uno de estos valores de ID.

Next, we loop through the list of users and print each user's username and ID ❸ . When Django asks which user to associate the existing topics with, we'll use one of these ID values.

Migrar la base de datos (Migrating the Database)

Ahora que conocemos los ID, podemos migrar la base de datos. Cuando lo hagamos, Python nos pedirá que conectemos el modelo `Topic` a un propietario concreto temporalmente o que añadamos un valor por defecto a nuestro archivo `models.py` para indicarle qué hacer. Elige la opción **1** :

Now that we know the IDs, we can migrate the database. When we do this, Python will ask us to connect the `Topic` model to a particular owner temporarily or to add a default to our `models.py` file to tell it what to do. Choose option **1** :

❶ (ll_env)learning_log\$ python manage.py makemigrations learning_logs

❷ It is impossible to add a non-nullable field 'owner' to topic without specifying a default. This is because...

❸ Please select a fix:

1) Provide a one-off default now (will be set on all existing rows with a null value for this column)

2) Quit and manually define a default value in models.py.

❹ Select an option: 1

❺ Please enter the default value now, as valid Python

The datetime and django.utils.timezone modules are available...

Type 'exit' to exit this prompt

❻ >>> 1

Migrations for 'learning_logs':

learning_logs/migrations/0003_topic_owner.py

- Add field owner to topic

(ll_env)learning_log\$

Comenzamos emitiendo el comando `makemigrations` **❶** . En la salida, Django indica que estamos intentando añadir un campo obligatorio (*non-nullable*) a un modelo existente (`topic`) sin valor por defecto especificado **❷** . Django nos da dos opciones: podemos proporcionar un valor por defecto ahora mismo, o podemos abandonar y añadir un valor por defecto en `models.py` **❸** . Aquí he elegido la primera opción **❹** . A continuación, Django nos pide que introduzcamos el valor por defecto **❺** .

We start by issuing the `makemigrations` command **❶** . In the output, Django indicates that we're trying to add a required (*non-nullable*) field to an existing model (`topic`) with no default value specified **❷** . Django gives us two options: we can provide a default right now, or we can quit and add a default value in `models.py` **❸** . Here I've chosen the first option **❹** . Django then asks us to enter the default value **❺** .

Para asociar todos los temas existentes al usuario administrador original, `ll_admin`, he introducido el ID de usuario de 1 ❹. Puedes utilizar el ID de cualquier usuario que hayas creado; no tiene por qué ser un superusuario. A continuación, Django migra la base de datos utilizando este valor y genera el archivo de migración `0003_topic_owner.py`, que añade el campo `owner` al modelo `Topic`.

To associate all existing topics with the original admin user, `ll_admin`, I entered the user ID of 1 ❹. You can use the ID of any user you've created; it doesn't have to be a superuser. Django then migrates the database using this value and generates the migration file `0003_topic_owner.py`, which adds the field `owner` to the `Topic` model.

Ahora podemos ejecutar la migración. Introduce lo siguiente en un entorno virtual activo:

Now we can execute the migration. Enter the following in an active virtual environment:

```
(ll_env)learning_log$ python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, learning_logs, sessions
Running migrations:
❶ Applying learning_logs.0003_topic_owner... OK
(ll_env)learning_log$
```

Django aplica la nueva migración, y el resultado es OK ❶.

Django applies the new migration, and the result is OK ❶.

Podemos verificar que la migración ha funcionado como se esperaba en una sesión de shell, de la siguiente forma:

We can verify that the migration worked as expected in a shell session, like this:

```
>>> from learning_logs.models import Topic
>>> for topic in Topic.objects.all():
...     print(topic, topic.owner)
...
Chess ll_admin
Rock Climbing ll_admin
>>>
```

Importamos `Topic` de `learning_logs.models` y luego recorremos todos los temas existentes, imprimiendo cada tema y el usuario al que pertenece. Puedes ver que cada tema pertenece ahora al usuario `ll_admin` . (Si obtienes un error al ejecutar este código, intenta salir de la shell e iniciar una nueva shell).

We import `Topic` from `learning_logs.models` and then loop through all existing topics, printing each topic and the user it belongs to. You can see that each topic now belongs to the user `ll_admin` . (If you get an error when you run this code, try exiting the shell and starting a new shell.)

Nota (Note)

Puedes simplemente reiniciar la base de datos en lugar de migrar, pero así perderás todos los datos existentes. Es una buena práctica aprender a migrar una base de datos manteniendo la integridad de los datos de los usuarios. Si quieres empezar con una base de datos nueva, ejecuta el comando `python manage.py flush` para reconstruir la estructura de la base de datos. Tendrás que crear un nuevo superusuario, y todos tus datos habrán desaparecido.

You can simply reset the database instead of migrating, but that will lose all existing data. It's good practice to learn how to migrate a database while maintaining the integrity of users' data. If you do want to start with a fresh database, issue the command `python manage.py flush` to rebuild the database structure. You'll have to create a new superuser, and all of your data will be gone.

Restringir el acceso a los temas a los usuarios adecuados (*Restricting Topics Access to Appropriate Users*)

Actualmente, si estás conectado, podrás ver todos los temas, independientemente del usuario con el que estés conectado. Cambiaremos eso mostrando a los usuarios sólo los temas que les pertenecen.

Currently, if you're logged in, you'll be able to see all the topics, no matter which user you're logged in as. We'll change that by showing users only the topics that belong to them.

Haz el siguiente cambio en la función `topics()` en `views.py` :

Make the following change to the `topics()` function in `views.py` :

learning_logs/views.py

learning_logs/views.py

```
--snip--
@login_required
def topics(request):
    """Show all topics."""
    topics = Topic.objects.filter(owner=request.user).order_by('date_added')
    context = {'topics': topics}
    return render(request, 'learning_logs/topics.xhtml', context)
--snip--
```

Cuando un usuario inicia sesión, el objeto `request` tiene un atributo `request.user` establecido, que contiene información sobre el usuario. La consulta `Topic.objects.filter(owner=request.user)` indica a Django que recupere sólo los objetos `Topic` de la base de datos cuyo atributo `owner` coincida con el usuario actual. Como no vamos a cambiar la forma en que se muestran los temas, no necesitamos cambiar en absoluto la plantilla de la página de temas.

When a user is logged in, the `request` object has a `request.user` attribute set, which contains information about the user. The query `Topic.objects.filter(owner=request.user)` tells Django to retrieve only the `Topic` objects from the database whose `owner` attribute matches the current user. Because we're not changing how the topics are displayed, we don't need to change the template for the topics page at all.

Para ver si esto funciona, inicia sesión como el usuario al que conectaste todos los temas existentes, y ve a la página de temas. Deberías ver todos los temas. Ahora cierra la sesión y vuelve a entrar como un usuario diferente. Deberías ver el mensaje "Aún no se ha añadido ningún tema".

To see if this works, log in as the user you connected all existing topics to, and go to the topics page. You should see all the topics. Now log out and log back in as a different user. You should see the message "No topics have been added yet."

Proteger los temas de un usuario (*Protecting a User's Topics*)

Aún no hemos restringido el acceso a las páginas de temas, por lo que cualquier usuario registrado podría probar un montón de URLs (como `http://localhost:8000/topics/1/`) y recuperar las páginas de temas que coincidan.

We haven't restricted access to the topic pages yet, so any registered user could try a bunch of URLs (like `http://localhost:8000/topics/1/`) and retrieve topic pages that happen to match.

Pruébalo tú mismo. Mientras estás conectado como el usuario propietario de todos los temas, copia la URL o anota el ID en la URL de un tema, y luego desconéctate y vuelve a conectarte como un usuario diferente. Introduce la URL de ese tema. Deberías poder leer las entradas, aunque hayas iniciado sesión como un usuario diferente.

Try it yourself. While logged in as the user that owns all topics, copy the URL or note the ID in the URL of a topic, and then log out and log back in as a different user. Enter that topic's URL. You should be able to read the entries, even though you're logged in as a different user.

Arreglaremos esto ahora realizando una comprobación antes de recuperar las entradas solicitadas en la función de vista `topic()` :

We'll fix this now by performing a check before retrieving the requested entries in the `topic()` view function:

learning_logs/views.py

learning_logs/views.py

```
from django.shortcuts import render, redirect
from django.contrib.auth.decorators import login_required
❶ from django.http import Http404

--snip--
@login_required
def topic(request, topic_id):
    """Show a single topic and all its entries."""
    topic = Topic.objects.get(id=topic_id)
    # Make sure the topic belongs to the current user.
    ❷ if topic.owner != request.user:
        raise Http404

    entries = topic.entry_set.order_by('-date_added')
    context = {'topic': topic, 'entries': entries}
    return render(request, 'learning_logs/topic.xhtml', context)
--snip--
```

Una respuesta 404 es una respuesta de error estándar que se devuelve cuando un recurso solicitado no existe en un servidor. Aquí importamos la excepción `Http404` ❶ , que lanzaremos si el usuario solicita un tema al que no debería tener acceso. Tras recibir una solicitud de tema, nos aseguramos de que el usuario del tema coincide con el usuario conectado en ese momento antes de mostrar la página. Si el propietario del tema solicitado no coincide con el usuario actual, lanzamos la excepción `Http404` ❷ , y Django devuelve una página de error 404.

A 404 response is a standard error response that's returned when a requested resource doesn't exist on a server. Here we import the `Http404` exception ❶, which we'll raise if the user requests a topic they shouldn't have access to. After receiving a topic request, we make sure the topic's user matches the currently logged-in user before rendering the page. If the requested topic's owner is not the same as the current user, we raise the `Http404` exception ❷, and Django returns a 404-error page.

Ahora, si intentas ver las entradas de los temas de otro usuario, verás un mensaje de "Página no encontrada" de Django. En el Capítulo 20, configuraremos el proyecto para que los usuarios vean una página de error adecuada en lugar de una página de depuración.

Now if you try to view another user's topic entries, you'll see a "Page Not Found" message from Django. In Chapter 20, we'll configure the project so users will see a proper error page instead of a debugging page.

Proteger la página `edit_entry` (*Protecting the `edit_entry` Page*)

Las páginas `edit_entry` tienen URLs de la forma `http://localhost:8000/edit_entry/entry_id/`, donde `entry_id` es un número. Protejamos esta página para que nadie pueda utilizar la URL para acceder a las entradas de otra persona:

The `edit_entry` pages have URLs of the form `http://localhost:8000/edit_entry/entry_id/`, where the `entry_id` is a number. Let's protect this page so no one can use the URL to gain access to someone else's entries:

learning_logs/views.py

learning_logs/views.py

```
--snip--
@login_required
def edit_entry(request, entry_id):
    """Edit an existing entry."""
    entry = Entry.objects.get(id=entry_id)
```

```
topic = entry.topic
if topic.owner != request.user:
    raise Http404

if request.method != 'POST':
    --snip--
```

Recuperamos la entrada y el tema asociado a esta entrada. A continuación, comprobamos si el propietario del tema coincide con el usuario conectado en ese momento; si no coinciden, lanzamos una excepción `Http404` .

We retrieve the entry and the topic associated with this entry. We then check whether the owner of the topic matches the currently logged-in user; if they don't match, we raise an `Http404` exception.

Asociar nuevos temas al usuario actual *(Associating New Topics with the Current User)*

Actualmente, la página para añadir nuevos temas está rota porque no asocia los nuevos temas a ningún usuario en particular. Si intentas añadir un nuevo tema, verás el mensaje `IntegrityError` junto con `NOT NULL constraint failed: learning_logs_topic.owner_id` . Django está diciendo que no puedes crear un nuevo tema sin especificar un valor para el campo `owner` del tema.

Currently, the page for adding new topics is broken because it doesn't associate new topics with any particular user. If you try adding a new topic, you'll see the message `IntegrityError` along with `NOT NULL constraint failed: learning_logs_topic.owner_id` . Django is saying you can't create a new topic without specifying a value for the topic's `owner` field.

Hay una solución sencilla para este problema, porque tenemos acceso al usuario actual a través del objeto `request` . Añade el siguiente código, que asocia el nuevo tema con el usuario actual:

There's a straightforward fix for this problem, because we have access to the current user through the `request` object. Add the following code, which associates the new topic with the current user:

learning_logs/views.py

learning_logs/views.py

```
--snip--
@login_required
def new_topic(request):
    --snip--
    else:
        # POST data submitted; process data.
        form = TopicForm(data=request.POST)
        if form.is_valid():
            ❶ new_topic = form.save(commit=False)
            ❷ new_topic.owner = request.user
            ❸ new_topic.save()
            return redirect('learning_logs:topics')

        # Display a blank or invalid form.
        context = {'form': form}
        return render(request, 'learning_logs/new_topic.xhtml', context)
--snip--
```

Quando llamamos por primera vez a `form.save()` , pasamos el argumento `commit=False` porque necesitamos modificar el nuevo tema antes de guardarlo en la base de datos ❶ . A continuación, establecemos el atributo `owner` del nuevo tema en el usuario actual ❷ . Por último, llamamos a `save()` en el tema que acabamos de definir ❸ . Ahora el tema tiene todos los datos necesarios y se guardará correctamente.

When we first call `form.save()` , we pass the `commit=False` argument because we need to modify the new topic before saving it to the database ❶ . We then set the new topic's `owner` attribute to the current user ❷ . Finally, we call `save()` on the topic instance we just defined ❸ . Now the topic has all the required data and will save successfully.

Deberías poder añadir tantos temas nuevos como quieras para tantos usuarios diferentes como desees. Cada usuario sólo tendrá acceso a sus propios datos, ya sea para verlos, introducir datos nuevos o modificar datos antiguos.

You should be able to add as many new topics as you want for as many different users as you want. Each user will only have access to their own data, whether they're viewing data, entering new data, or modifying old data.

Pruébalo tú mismo (*Try It Yourself*)

19-3. Refactorización: Hay dos lugares en *views.py* en los que nos aseguramos de que el usuario asociado a un tema coincide con el usuario conectado en ese momento. Pon el código para esta comprobación en una función llamada `check_topic_owner()` , y llama a esta función cuando sea apropiado.

19-3. Refactoring: There are two places in *views.py* where we make sure the user associated with a topic matches the currently logged-in user. Put the code for this check in a function called `check_topic_owner()` , and call this function where appropriate.

19-4. Proteger nueva_entrada: Actualmente, un usuario puede añadir una nueva entrada al registro de aprendizaje de otro usuario introduciendo una URL con el ID de un tema perteneciente a otro usuario. Evita este ataque comprobando que el usuario actual es el propietario del tema de la entrada antes de guardar la nueva entrada.

19-4. Protecting new_entry: Currently, a user can add a new entry to another user's learning log by entering a URL with the ID of a topic belonging to another user. Prevent this attack by checking that the current user owns the entry's topic before saving the new entry.

19-5. Blog protegido: En tu proyecto de Blog, asegúrate de que cada entrada del blog está conectada a un usuario concreto. Asegúrate de que todas las entradas son accesibles públicamente, pero sólo los usuarios registrados pueden añadir entradas y editar las existentes. En la vista que permite a los usuarios editar sus entradas, asegúrate de que el usuario está editando su propia entrada antes de procesar el formulario.

19-5. Protected Blog: In your Blog project, make sure each blog post is connected to a particular user. Make sure all posts are publicly accessible but only registered users can add posts and edit existing posts. In the view that allows users to edit their posts, make sure the user is editing their own post before processing the form.

Resumen (Summary)

En este capítulo, has aprendido cómo los formularios permiten a los usuarios añadir nuevos temas y entradas, y editar las entradas existentes. Luego aprendiste a implementar cuentas de usuario. Proporcionaste a los usuarios existentes la capacidad de iniciar y cerrar sesión, y utilizaste la dirección `UserCreationForm` predeterminada de Django para permitir a la gente crear nuevas cuentas.

In this chapter, you learned how forms allow users to add new topics and entries, and edit existing entries. You then learned how to implement user accounts. You gave existing users the ability to log in and out, and used Django's default `UserCreationForm` to let people create new accounts.

Después de construir un sistema sencillo de autenticación y registro de usuarios, restringiste el acceso a los usuarios registrados para determinadas páginas utilizando el decorador `@login_required`. A continuación, asignaste datos a usuarios específicos mediante una relación de clave foránea. También aprendiste a migrar la base de datos cuando la migración requiere que especifiques algunos datos por defecto.

After building a simple user authentication and registration system, you restricted access to logged-in users for certain pages using the `@login_required` decorator. You then assigned data to specific users through a foreign key relationship. You also learned to migrate the database when the migration requires you to specify some default data.

Por último, aprendiste a asegurarte de que un usuario sólo puede ver los datos que le pertenecen modificando las funciones de vista. Recuperaste los datos apropiados utilizando el método `filter()`, y comparaste el propietario de los datos solicitados con el usuario actualmente conectado.

Finally, you learned how to make sure a user can only see data that belongs to them by modifying the view functions. You retrieved appropriate data using the `filter()` method, and compared the owner of the requested data to the currently logged-in user.

Puede que no siempre sea inmediatamente obvio qué datos debes poner a disposición y qué datos debes proteger, pero esta habilidad se adquiere con la práctica. Las decisiones que hemos tomado en este capítulo para proteger los datos de nuestros usuarios también ilustran por qué trabajar con otros es una buena idea cuando se construye un proyecto: que otra persona revise tu proyecto hace que sea más probable que detectes las áreas vulnerables.

It might not always be immediately obvious what data you should make available and what data you should protect, but this skill will come with practice. The decisions we've made in this chapter to secure our users' data also illustrate why working with others is a good idea when building a project: having someone else look over your project makes it more likely that you'll spot vulnerable areas.

Ahora tienes un proyecto totalmente operativo en tu máquina local. En el último capítulo, darás estilo al Registro de Aprendizaje para hacerlo visualmente atractivo, y desplegarás el proyecto en un servidor para que cualquier persona con acceso a Internet pueda registrarse y crear una cuenta.

You now have a fully functioning project running on your local machine. In the final chapter, you'll style Learning Log to make it visually appealing, and you'll deploy the project to a server so anyone with internet access can register and make an account.

Capítulo 20: Estilizar e implementar una aplicación (Chapter 20: Styling and Deploying an App)

20

Estilizar e implementar una aplicación (20 Styling and Deploying an App)



DESCRIPCIÓN DE LA IMAGEN (En español):

La imagen muestra una ilustración de una serpiente, con una apariencia animada y dibujos detallados de sus escamas, que está rodeando un objeto cilíndrico que parece un motor a reacción. La serpiente tiene una expresión alerta y los ojos bien definidos. El dibujo utiliza tonos monocromáticos, principalmente blanco y negro, sobre un fondo gris.

Learning Log ya es totalmente funcional, pero no tiene estilo y sólo se ejecuta en tu máquina local. En este capítulo, estilizarás el proyecto de forma sencilla pero profesional y luego lo desplegarás en un servidor en vivo para que cualquier persona del mundo pueda crear una cuenta y utilizarlo.

Learning Log is fully functional now, but it has no styling and runs only on your local machine. In this chapter, you'll style the project in a simple but professional manner and then deploy it to a live server so anyone in the world can make an account and use it.

Para el estilo, utilizaremos la biblioteca *Bootstrap*, una colección de herramientas para dar estilo a las aplicaciones web de modo que tengan un aspecto profesional en todos los dispositivos modernos, desde un pequeño teléfono a un gran monitor de escritorio. Para ello, utilizaremos la aplicación *django-bootstrap5*, que también te servirá para practicar el uso de aplicaciones creadas por otros desarrolladores de Django.

For the styling, we'll use the *Bootstrap* library, a collection of tools for styling web applications so they look professional on all modern devices, from a small phone to a large desktop monitor. To do this, we'll use the `django-bootstrap5` app, which will also give you practice using apps made by other Django developers.

Desplegaremos Learning Log utilizando *Platform.sh*, un sitio que te permite enviar tu proyecto a uno de sus servidores, poniéndolo a disposición de cualquier persona con conexión a Internet. También empezaremos a utilizar un sistema de control de versiones llamado Git para realizar un seguimiento de los cambios en el proyecto.

We'll deploy Learning Log using *Platform.sh*, a site that lets you push your project to one of its servers, making it available to anyone with an internet connection. We'll also start using a version control system called Git to track changes to the project.

Cuando hayas terminado con Learning Log, serás capaz de desarrollar aplicaciones web sencillas, darles un aspecto profesional y desplegarlas en un servidor activo. También podrás utilizar recursos de aprendizaje más avanzados a medida que desarrolles tus habilidades.

When you're finished with Learning Log, you'll be able to develop simple web applications, give them a professional look and feel, and deploy them to a live server. You'll also be able to use more advanced learning resources as you develop your skills.

Estilo de Learning Log (*Styling Learning Log*)

Hemos ignorado a propósito el estilo hasta ahora para centrarnos primero en la funcionalidad de Learning Log. Es una buena forma de enfocar el desarrollo, porque una aplicación sólo es útil si funciona. Una vez que la aplicación funciona, su aspecto es fundamental para que la gente quiera utilizarla.

We've purposely ignored styling until now to focus on Learning Log's functionality first. This is a good way to approach development, because an app is only useful if it works. Once an app is working, its appearance is critical so people will want to use it.

En esta sección, instalaremos la aplicación django-bootstrap5 y la añadiremos al proyecto. Luego la utilizaremos para dar estilo a las páginas individuales del proyecto, de modo que todas las páginas tengan un aspecto coherente.

In this section, we'll install the django-bootstrap5 app and add it to the project. We'll then use it to style the individual pages in the project, so all the pages have a consistent look and feel.

La aplicación django-bootstrap5 (*The django-bootstrap5 App*)

Utilizaremos django-bootstrap5 para integrar Bootstrap en nuestro proyecto. Esta aplicación descarga los archivos de Bootstrap necesarios, los coloca en una ubicación adecuada de tu proyecto y hace que las directivas de estilo estén disponibles en las plantillas de tu proyecto.

We'll use django-bootstrap5 to integrate Bootstrap into our project. This app downloads the required Bootstrap files, places them in an appropriate location in your project, and makes the styling directives available in your project's templates.

Para instalar django-bootstrap5, ejecuta el siguiente comando en un entorno virtual activo:

To install django-bootstrap5, issue the following command in an active virtual environment:

```
(ll_env)learning_log$ pip install django-bootstrap5
```

```
--snip--
```

```
Successfully installed beautifulsoup4-4.11.1 django-bootstrap5-21.3  
soupsieve-2.3.2.post1
```

A continuación, tenemos que añadir django-bootstrap5 a `INSTALLED_APPS` en `settings.py` :

Next, we need to add django-bootstrap5 to `INSTALLED_APPS` in `settings.py` :

settings.py

settings.py

```
--snip--
INSTALLED_APPS = [
    # My apps.
    'learning_logs',
    'accounts',

    # Third party apps.
    'django_bootstrap5',

    # Default django apps.
    'django.contrib.admin',
--snip--
```

Crea una nueva sección llamada `Third party apps` , para aplicaciones creadas por otros desarrolladores, y añade `'django_bootstrap5'` a esta sección. Asegúrate de colocar esta sección después de `My apps` pero antes de la sección que contiene las aplicaciones por defecto de Django.

Start a new section called `Third party apps` , for apps created by other developers, and add `'django_bootstrap5'` to this section. Make sure you place this section after `My apps` but before the section containing Django's default apps.

Utilizar Bootstrap para dar estilo al registro de aprendizaje (Using Bootstrap to Style Learning Log)

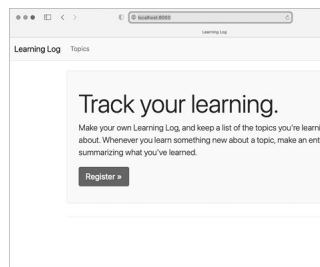
Bootstrap es una gran colección de herramientas de estilo. También tiene una serie de plantillas que puedes aplicar a tu proyecto para crear un estilo general. Es mucho más fácil utilizar estas plantillas que utilizar herramientas de estilo individuales. Para ver las plantillas que ofrece Bootstrap, ve a <https://getbootstrap.com> y haz clic en **Ejemplos** . Utilizaremos la plantilla *Navbar static* , que proporciona una sencilla barra de navegación superior y un contenedor para el contenido de la página.

Bootstrap is a large collection of styling tools. It also has a number of templates you can apply to your project to create an overall style. It's much easier to use these templates than

to use individual styling tools. To see the templates Bootstrap offers, go to <https://getbootstrap.com> and click **Examples**. We'll use the *Navbar static* template, which provides a simple top navigation bar and a container for the page's content.

La Figura 20-1 muestra el aspecto que tendrá la página de inicio después de que apliquemos la plantilla de Bootstrap a *base.xhtml* y modifiquemos ligeramente *index.xhtml*.

Figure 20-1 shows what the home page will look like after we apply Bootstrap's template to *base.xhtml* and modify *index.xhtml* slightly.



DESCRIPCIÓN DE LA
IMAGEN (En español):

Registro de
Aprendizaje Temas
Registrarse Iniciar
sesión Sigue tu
aprendizaje. Crea tu
propio Registro de
Aprendizaje y lleva una
lista de los temas que
estás aprendiendo.
Cada vez que aprendes
algo nuevo sobre un
tema, haz una
anotación que resuma
lo que has aprendido.
Registrarse »

Figura 20-1 : Página de inicio del Registro de aprendizaje utilizando Bootstrap

Figure 20-1 : The Learning Log home page using Bootstrap

Modificación de *base.xhtml* (*Modifying base.xhtml*)

Tenemos que reescribir *base.xhtml* utilizando la plantilla de Bootstrap. Desarrollaremos el nuevo *base.xhtml* por secciones. Se trata de un archivo grande; tal vez quieras copiarlo de los recursos en línea, disponibles en https://ehmatthes.github.io/pcc_3e. Si copias el archivo, deberías leer la siguiente sección para entender los cambios realizados.

We need to rewrite *base.xhtml* using the Bootstrap template. We'll develop the new *base.xhtml* in sections. This is a large file; you may want to copy this file from the online resources, available at https://ehmatthes.github.io/pcc_3e. If you do copy the file, you should still read through the following section to understand the changes that were made.

Definición de las cabeceras HTML (Defining the HTML Headers)

El primer cambio que haremos en *base.xhtml* define las cabeceras HTML del archivo. También añadiremos algunos requisitos para utilizar Bootstrap en nuestras plantillas, y daremos un título a la página. Elimina todo lo que hay en *base.xhtml* y sustitúyelo por el siguiente código:

The first change we'll make to *base.xhtml* defines the HTML headers in the file. We'll also add some requirements for using Bootstrap in our templates, and give the page a title. Delete everything in *base.xhtml* and replace it with the following code:

base.xhtml

base.xhtml

```
❶ <!doctype html>
❷ <html lang="en">
❸ <head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
❹ <title>Learning Log</title>

❺ {% load django_bootstrap5 %}
  {% bootstrap_css %}
  {% bootstrap_javascript %}

</head>
```

Primero declaramos este archivo como un documento HTML ❶ escrito en inglés ❷. Un archivo HTML se divide en dos partes principales: el encabezado *head* y el encabezado *body*. El encabezado del archivo comienza con una etiqueta de apertura `<head>` ❸. La cabecera de un archivo HTML no contiene nada del contenido de la página; sólo indica al navegador lo que necesita saber para mostrar la página correctamente. Incluimos un elemento `<title>` para la página, que se mostrará en la barra de título del navegador siempre que Learning Log esté abierto ❹.

We first declare this file as an HTML document ❶ written in English ❷ . An HTML file is divided into two main parts: the *head* and the *body* . The head of the file begins with an opening `<head>` tag ❸ . The head of an HTML file doesn't hold any of the page's content; it just tells the browser what it needs to know to display the page correctly. We include a `<title>` element for the page, which will display in the browser's title bar whenever Learning Log is open ❹ .

Antes de cerrar la sección head, cargamos la colección de etiquetas de plantilla disponibles en django-bootstrap5 ❺ . La etiqueta de plantilla `{% bootstrap_css %}` es una etiqueta personalizada de django-bootstrap5; carga todos los archivos CSS necesarios para implementar los estilos Bootstrap. La etiqueta que sigue habilita todo el comportamiento interactivo que podrías utilizar en una página, como las barras de navegación plegables. La etiqueta de cierre `</head>` aparece en la última línea.

Before closing the head section, we load the collection of template tags available in django-bootstrap5 ❺ . The template tag `{% bootstrap_css %}` is a custom tag from django-bootstrap5; it loads all of the CSS files required to implement Bootstrap styles. The tag that follows enables all the interactive behavior you might use on a page, such as collapsible navigation bars. The closing `</head>` tag appears on the last line.

Todas las opciones de estilo de Bootstrap están ahora disponibles en cualquier plantilla que herede de *base.xhtml* . Si quieres utilizar etiquetas de plantilla personalizadas de django-bootstrap5, cada plantilla deberá incluir la etiqueta `{% load django_bootstrap5 %}` .

All Bootstrap styling options are now available in any template that inherits from *base.xhtml* . If you want to use custom template tags from django-bootstrap5, each template will need to include the `{% load django_bootstrap5 %}` tag.

Definir la barra de navegación (Defining the Navigation Bar)

El código que define la barra de navegación en la parte superior de la página es bastante largo, porque tiene que funcionar igual de bien en las pantallas estrechas de los teléfonos y en las anchas de los monitores de escritorio. Trabajaremos la barra de navegación por secciones.

The code that defines the navigation bar at the top of the page is fairly long, because it has to work equally well on narrow phone screens and wide desktop monitors. We'll work through the navigation bar in sections.

Aquí está la primera parte de la barra de navegación:

Here's the first part of the navigation bar:

base.xhtml

base.xhtml

```
--snip--
</head>
<body>
```

```
❶ <nav class="navbar navbar-expand-md navbar-light bg-light mb-4 border">
  <div class="container-fluid">
❷    <a class="navbar-brand" href="{% url 'learning_logs:index' %}">
      Learning Log</a>

❸    <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
        data-bs-target="#navbarCollapse" aria-controls="navbarCollapse"
        aria-expanded="false" aria-label="Toggle navigation">
      <span class="navbar-toggler-icon"></span>
    </button>

❹    <div class="collapse navbar-collapse" id="navbarCollapse">
❺      <ul class="navbar-nav me-auto mb-2 mb-md-0">
❻        <li class="nav-item">
❼          <a class="nav-link" href="{% url 'learning_logs:topics' %}">
              Topics</a></li>
        </ul> <!-- End of links on left side of navbar -->
      </div> <!-- Closes collapsible parts of navbar -->

    </div> <!-- Closes navbar's container -->
  </nav> <!-- End of navbar -->
```

⑧ {% block content %}{% endblock content %}

</body>

</html>

El primer elemento nuevo es la etiqueta de apertura `<body>` . La etiqueta *body* de un archivo HTML contiene el contenido que los usuarios verán en una página. A continuación tenemos un elemento `<nav>` , que abre el código de la barra de navegación en la parte superior de la página ❶ . Todo lo que contiene este elemento está estilizado según las reglas de estilo de Bootstrap definidas por los selectores `navbar` , `navbar-expand-md` , y el resto que ves aquí. Un *selector* determina a qué elementos de una página se aplica una determinada regla de estilo. Los selectores `navbar-light` y `bg-light` dan estilo a la barra de navegación con un fondo de tema claro. El `mb` en `mb-4` es la abreviatura de *margin-bottom* ; este selector garantiza que aparezca un pequeño espacio entre la barra de navegación y el resto de la página. El selector `border` proporciona un fino borde alrededor del fondo claro para diferenciarlo un poco del resto de la página.

The first new element is the opening `<body>` tag. The *body* of an HTML file contains the content users will see on a page. Next we have a `<nav>` element, which opens the code for the navigation bar at the top of the page ❶ . Everything contained in this element is styled according to the Bootstrap style rules defined by the selectors `navbar` , `navbar-expand-md` , and the rest that you see here. A *selector* determines which elements on a page a certain style rule applies to. The `navbar-light` and `bg-light` selectors style the navigation bar with a light-themed background. The `mb` in `mb-4` is short for *margin-bottom* ; this selector ensures that a little space appears between the navigation bar and the rest of the page. The `border` selector provides a thin border around the light background to set it off a little from the rest of the page.

La etiqueta `<div>` de la línea siguiente abre un contenedor redimensionable que contendrá la barra de navegación general. El término *div* es la abreviatura de *division* ; construyes una página web dividiéndola en secciones y definiendo reglas de estilo y comportamiento que se aplican a cada sección. Cualquier regla de estilo o comportamiento que se defina en una etiqueta de apertura `<div>` afectará a todo lo que veas hasta su correspondiente etiqueta de cierre, escrita como `</div>` .

The `<div>` tag on the next line opens a resizable container that will hold the overall navigation bar. The term *div* is short for *division* ; you build a web page by dividing it into sections and defining style and behavior rules that apply to that section. Any styling or

behavior rules that are defined in an opening `<div>` tag affect everything you see until its corresponding closing tag, written as `</div>` .

A continuación establecemos el nombre del proyecto, `Learning Log` , para que aparezca como primer elemento de la barra de navegación ❷ . Esto también servirá como enlace a la página de inicio, tal y como ha estado haciendo en la versión mínimamente estilizada del proyecto que construimos en los dos capítulos anteriores. El selector `navbar-brand` da estilo a este enlace para que destaque del resto de enlaces y ayude a añadir algo de marca al sitio.

Next we set the project's name, `Learning Log` , to appear as the first element on the navigation bar ❷ . This will also serve as a link to the home page, just as it's been doing in the minimally styled version of the project we built in the previous two chapters. The `navbar-brand` selector styles this link so it stands out from the rest of the links and helps add some branding to the site.

A continuación, la plantilla Bootstrap define un botón que aparece si la ventana del navegador es demasiado estrecha para mostrar toda la barra de navegación horizontalmente ❸ . Cuando el usuario pulsa el botón, los elementos de navegación aparecen en una lista desplegable. La referencia `collapse` hace que la barra de navegación se colapse cuando el usuario encoge la ventana del navegador o cuando el sitio se muestra en dispositivos con pantallas pequeñas.

The Bootstrap template then defines a button that appears if the browser window is too narrow to display the whole navigation bar horizontally ❸ . When the user clicks the button, the navigation elements appear in a drop-down list. The `collapse` reference causes the navigation bar to collapse when the user shrinks the browser window or when the site is displayed on devices with small screens.

A continuación, abrimos una nueva sección (`<div>`) de la barra de navegación ❹ . Esta es la parte de la barra de navegación que puede colapsarse en función del tamaño de la ventana del navegador.

Next, we open a new section (`<div>`) of the navigation bar ❹ . This is the part of the navigation bar that can collapse depending on the size of the browser window.

Bootstrap define los elementos de navegación como elementos de una lista desordenada ⑤ , con reglas de estilo que hacen que no se parezca en nada a una lista. Todos los enlaces o elementos que necesites en la barra pueden incluirse como elementos de una lista desordenada ⑥ . Aquí, el único elemento de la lista es nuestro enlace a la página de temas ⑦ . Fíjate en la etiqueta de cierre `</li>` al final del enlace; toda etiqueta de apertura necesita su correspondiente etiqueta de cierre.

Bootstrap defines navigation elements as items in an unordered list ⑤ , with style rules that make it look nothing like a list. Every link or element you need on the bar can be included as an item in an unordered list ⑥ . Here, the only item in the list is our link to the topics page ⑦ . Notice the closing `</li>` tag at the end of the link; every opening tag needs a corresponding closing tag.

El resto de las líneas que se muestran aquí cierran todas las etiquetas que se han abierto. En HTML, un comentario se escribe así:

The rest of the lines shown here close out all of the tags that have been opened. In HTML, a comment is written like this:

```
<!-- This is an HTML comment. -->
```

Las etiquetas de cierre no suelen tener comentarios, pero si eres nuevo en HTML, puede ser realmente útil etiquetar algunas de tus etiquetas de cierre. Una sola etiqueta que falte o una etiqueta de más puede desvirtuar el diseño de toda una página. Incluimos el bloque `content` ⑧ y también las etiquetas de cierre `</body>` y `</html>` .

Closing tags don't usually have comments, but if you're new to HTML, it can be really helpful to label some of your closing tags. A single missing tag or an extra tag can throw off the layout of an entire page. We include the `content` block ⑧ and the closing `</body>` and `</html>` tags as well.

No hemos terminado con la barra de navegación, pero ahora tenemos un documento HTML completo. Si `runserver` está actualmente activo, detén el servidor actual y reinícialo. Ve a la página de inicio del proyecto, y deberías ver una barra de navegación que tiene algunos de los elementos mostrados en [la Figura 20-1](#) . Ahora vamos a añadir el resto de elementos

a la barra de navegación.

We're not finished with the navigation bar, but we now have a complete HTML document. If `runserver` is currently active, stop the current server and restart it. Go to the project's home page, and you should see a navigation bar that has some of the elements shown in [Figure 20-1](#). Now let's add the rest of the elements to the navigation bar.

Añadir enlaces a cuentas de usuario (Adding User Account Links)

Todavía tenemos que añadir los enlaces asociados a las cuentas de usuario. Empezaremos por añadir todos los enlaces relacionados con las cuentas, excepto el formulario de cierre de sesión.

We still need to add the links associated with user accounts. We'll start by adding all of the account-related links except the logout form.

Haz los siguientes cambios en *base.xhtml* :

Make the following changes to *base.xhtml* :

base.xhtml

base.xhtml

```
--snip--
</ul> <!-- End of links on left side of navbar -->

<!-- Account-related links -->
❶ <ul class="navbar-nav ms-auto mb-2 mb-md-0">

❷   {% if user.is_authenticated %}
   <li class="nav-item">
❸     <span class="navbar-text me-2">Hello, {{ user.username }}.
```

```

    </span></li>
4    {% else %}
    <li class="nav-item">
      <a class="nav-link" href="{% url 'accounts:register' %}">
        Register</a></li>
    <li class="nav-item">
      <a class="nav-link" href="{% url 'accounts:login' %}">
        Log in</a></li>
    {% endif %}

</ul> <!-- End of account-related links -->

</div> <!-- Closes collapsible parts of navbar -->
--snip--

```

Comenzamos un nuevo grupo de enlaces utilizando otra etiqueta de apertura `<ul>` ❶ . Puedes tener tantos grupos de enlaces como necesites en una página. El selector `ms-auto` es la abreviatura de *margin-start-automatic* : este selector examina los demás elementos de la barra de navegación y elabora un margen izquierdo (de inicio) que empuja este grupo de enlaces hacia el lado derecho de la ventana del navegador.

We begin a new set of links by using another opening `<ul>` tag ❶ . You can have as many groups of links as you need on a page. The selector `ms-auto` is short for *margin-start-automatic* : this selector examines the other elements in the navigation bar and works out a left (start) margin that pushes this group of links to the right side of the browser window.

El bloque `if` es el mismo bloque condicional que utilizamos antes para mostrar mensajes adecuados a los usuarios, dependiendo de si han iniciado sesión ❷ . Ahora el bloque es un poco más largo porque hay algunas reglas de estilo dentro de las etiquetas condicionales. El saludo para los usuarios autenticados se envuelve en un elemento `<span>` ❸ . Un *span element* estiliza trozos de texto o elementos de una página que forman parte de una línea más larga. Mientras que los elementos `div` crean sus propias divisiones en una página, los elementos `span` son continuos dentro de una sección mayor. Esto puede resultar confuso al principio, porque muchas páginas tienen elementos `div` profundamente anidados. Aquí, estamos utilizando el elemento `span` para dar estilo al texto informativo de la barra de navegación: en este caso, el nombre del usuario conectado.

The `if` block is the same conditional block we used earlier to display appropriate messages to users, depending on whether they're logged in ❷ . The block is a little longer now because there are some styling rules inside the conditional tags. The greeting for

authenticated users is wrapped in a `<span>` element ❸ . A *span element* styles pieces of text or elements of a page that are part of a longer line. While `div` elements create their own divisions in a page, `span` elements are continuous within a larger section. This can be confusing at first, because many pages have deeply nested `div` elements. Here, we're using the `span` element to style informational text on the navigation bar: in this case, the logged-in user's name.

En el bloque `else` , que se ejecuta para los usuarios no autenticados, incluimos los enlaces para registrar una nueva cuenta e iniciar sesión ❹ . Deben tener el mismo aspecto que el enlace a la página de temas.

In the `else` block, which runs for unauthenticated users, we include the links for registering a new account and logging in ❹ . These should look just like the link to the topics page.

Si quisieras añadir más enlaces a la barra de navegación, añadirías otro elemento `<li>` a uno de los grupos `<ul>` que hemos definido, utilizando directivas de estilo como las que has visto aquí.

If you wanted to add more links to the navigation bar, you'd add another `<li>` item to one of the `<ul>` groups that we've defined, using styling directives like the ones you've seen here.

Ahora vamos a añadir el formulario de cierre de sesión a la barra de navegación.

Now let's add the logout form to the navigation bar.

Añadir el formulario de cierre de sesión a la barra de navegación (Adding the Logout Form to the Navigation Bar)

Cuando escribimos por primera vez el formulario de cierre de sesión, lo añadimos a la parte inferior de *base.xhtml* . Ahora vamos a ponerlo en un lugar mejor, en la barra de navegación:

When we first wrote the logout form, we added it to the bottom of *base.xhtml* . Now let's put it in a better place, in the navigation bar:

base.xhtml

base.xhtml

```
--snip--
</ul> <!-- End of account-related links -->

{% if user.is_authenticated %}
  <form action="{% url 'accounts:logout' %}" method='post'>
    {% csrf_token %}
    ❶   <button name='submit' class='btn btn-outline-secondary btn-sm'>
      Log out</button>
    </form>
  {% endif %}

</div> <!-- Closes collapsible parts of navbar -->
--snip--
```

El formulario de cierre de sesión debe colocarse después del conjunto de enlaces relacionados con la cuenta, pero dentro de la sección plegable de la barra de navegación. El único cambio en el formulario es la adición de varias clases de estilo Bootstrap en el elemento `<button>` , que aplican elementos de estilo Bootstrap al botón de cierre de sesión ❶ .

The logout form should be placed after the set of account-related links, but inside the collapsible section of the navigation bar. The only change in the form is the addition of a number of Bootstrap styling classes in the `<button>` element, which apply Bootstrap styling elements to the logout button ❶ .

Recarga la página de inicio y podrás iniciar y cerrar sesión con cualquiera de las cuentas que hayas creado.

Reload the home page, and you should be able to log in and out using any of the accounts you've created.

Aún nos queda algo más que añadir a *base.xhtml*. Tenemos que definir dos bloques que las páginas individuales pueden utilizar para colocar el contenido específico de esas páginas.

There's still a bit more we need to add to *base.xhtml*. We need to define two blocks that the individual pages can use to place the content specific to those pages.

Definir la parte principal de la página (Defining the Main Part of the Page)

El resto de *base.xhtml* contiene la parte principal de la página:

The rest of *base.xhtml* contains the main part of the page:

base.xhtml

base.xhtml

```
--snip--
</nav> <!-- End of navbar -->

❶ <main class="container">
❷ <div class="pb-2 mb-2 border-bottom">
    {% block page_header %}{% endblock page_header %}
</div>
❸ <div>
    {% block content %}{% endblock content %}
</div>
</main>

</body>
</html>
```

Primero abrimos una etiqueta `<main>` ❶ . El elemento *main* se utiliza para la parte más significativa del cuerpo de una página. Aquí asignamos el selector bootstrap `container` , que es una forma sencilla de agrupar elementos en una página. Colocaremos dos elementos `div` en este contenedor.

We first open a `<main>` tag ❶ . The *main* element is used for the most significant part of the body of a page. Here we assign the bootstrap selector `container` , which is a simple way to group elements on a page. We'll place two `div` elements in this container.

El primer elemento `div` contiene un bloque `page_header` ❷ . Utilizaremos este bloque para titular la mayoría de las páginas. Para que esta sección destaque del resto de la página, colocaremos algo de relleno debajo del encabezamiento. *Padding* se refiere al espacio entre el contenido de un elemento y su borde. El selector `pb-2` es una directiva de bootstrap que proporciona una cantidad moderada de relleno en la parte inferior del elemento estilizado. Un *margin* es el espacio entre el borde de un elemento y otros elementos de la página. El selector `mb-2` proporciona una cantidad moderada de margen en la parte inferior de este `div`. Queremos un borde en la parte inferior de este bloque, así que utilizamos el selector `border-bottom` , que proporciona un borde fino en la parte inferior del bloque `page_header` .

The first `div` element contains a `page_header` block ❷ . We'll use this block to title most pages. To make this section stand out from the rest of the page, we place some padding below the header. *Padding* refers to space between an element's content and its border. The selector `pb-2` is a bootstrap directive that provides a moderate amount of padding at the bottom of the styled element. A *margin* is the space between an element's border and other elements on the page. The selector `mb-2` provides a moderate amount of margin at the bottom of this `div`. We want a border on the bottom of this block, so we use the selector `border-bottom` , which provides a thin border at the bottom of the `page_header` block.

A continuación, definimos un elemento `div` más que contiene el bloque `content` ❸ . No aplicamos ningún estilo específico a este bloque, por lo que podemos dar estilo al contenido de cualquier página como creamos conveniente para esa página. El final del archivo *base.xhtml* tiene etiquetas de cierre para los elementos `main` , `body` y `html` .

We then define one more `div` element that contains the block `content` ❸ . We don't apply any specific style to this block, so we can style the content of any page as we see fit for that

page. The end of the *base.xhtml* file has closing tags for the `main` , `body` , and `html` elements.

Cuando cargues la página de inicio de Learning Log en un navegador, deberías ver una barra de navegación de aspecto profesional que coincida con la que se muestra en [la Figura 20-1](#) . Prueba a cambiar el tamaño de la ventana para que sea muy estrecha; un botón debería sustituir a la barra de navegación. Haz clic en el botón, y todos los enlaces deberían aparecer en una lista desplegable.

When you load Learning Log’s home page in a browser, you should see a professional-looking navigation bar that matches the one shown in [Figure 20-1](#) . Try resizing the window so it’s really narrow; a button should replace the navigation bar. Click the button, and all the links should appear in a drop-down list.

Estilizar la página de inicio con una jumbotrona (*Styling the Home Page Using a Jumbotron*)

Para actualizar la página de inicio, utilizaremos un elemento de Bootstrap llamado *jumbotron* , una gran caja que destaca del resto de la página. Normalmente, se utiliza en las páginas de inicio para contener una breve descripción del proyecto general y una llamada a la acción que invite al espectador a participar.

To update the home page, we’ll use a Bootstrap element called a *jumbotron* , a large box that stands out from the rest of the page. Typically, it’s used on home pages to hold a brief description of the overall project and a call to action that invites the viewer to get involved.

Aquí tienes el archivo *index.xhtml* revisado:

Here’s the revised *index.xhtml* file:

index.xhtml

index.xhtml

```
{% extends "learning_logs/base.xhtml" %}

❶ {% block page_header %}
❷ <div class="p-3 mb-4 bg-light border rounded-3">
  <div class="container-fluid py-4">
❸    <h1 class="display-3">Track your learning.</h1>

❹    <p class="lead">Make your own Learning Log, and keep a list of the
      topics you're learning about. Whenever you learn something new
      about a topic, make an entry summarizing what you've learned.</p>

❺    <a class="btn btn-primary btn-lg mt-1"
      href="{% url 'accounts:register' %}">Register &raquo;</a>
  </div>
</div>
{% endblock page_header %}
```

Primero le decimos a Django que vamos a definir lo que va en el bloque `page_header` ❶ . Una jumbotron se implementa como un par de elementos `div` con un conjunto de directivas de estilo aplicadas a ellos ❷ . El `div` exterior tiene ajustes de relleno y margen, un color de fondo claro y esquinas redondeadas. El `div` interior es un contenedor que cambia con el tamaño de la ventana y también tiene algo de relleno. El selector `py-4` añade relleno a la parte superior e inferior del elemento `div`. Siéntete libre de ajustar los números de estas opciones y observa cómo cambia la página de inicio.

We first tell Django that we're about to define what goes in the `page_header` block ❶ . A jumbotron is implemented as a pair of `div` elements with a set of styling directives applied to them ❷ . The outer `div` has padding and margin settings, a light background color, and rounded corners. The inner `div` is a container that changes along with the window size and has some padding as well. The `py-4` selector adds padding to the top and bottom of the `div` element. Feel free to adjust the numbers in these settings and see how the home page changes.

Dentro del jumbotrón hay tres elementos. El primero es un breve mensaje, `Track your learning` , que da a los nuevos visitantes una idea de lo que hace Learning Log ❸ . El elemento `<h1>` es un encabezado de primer nivel, y el selector `display-3` añade un aspecto más fino y alto a este encabezado en particular. También incluimos un mensaje más largo que proporciona más información sobre lo que el usuario puede hacer con su registro de aprendizaje ❹ . Está formateado como un párrafo `lead` , que pretende destacarse de los párrafos normales.

Inside the jumbotron are three elements. The first is a short message, Track your learning , that gives new visitors a sense of what Learning Log does ❸ . The `<h1>` element is a first-level header, and the `display-3` selector adds a thinner and taller look to this particular header. We also include a longer message that provides more information about what the user can do with their learning log ❹ . This is formatted as a `lead` paragraph, which is meant to stand out from regular paragraphs.

En lugar de utilizar sólo un enlace de texto, creamos un botón que invita a los usuarios a registrar una cuenta en Learning Log ❺ . Se trata del mismo enlace que en la cabecera, pero el botón destaca en la página y muestra al espectador lo que tiene que hacer para empezar a utilizar el proyecto. Los selectores que ves aquí estilizan esto como un botón grande que representa una llamada a la acción. El código `»` es un *HTML entity* que se parece a dos corchetes combinados (`>>`). Por último, proporcionamos etiquetas `div` de cierre y cerramos el bloque `page_header` . Con sólo dos elementos `div` en este archivo, no es especialmente útil etiquetar las etiquetas `div` de cierre. No vamos a añadir nada más a esta página, así que no necesitamos definir el bloque `content` en esta plantilla.

Rather than just using a text link, we create a button that invites users to register an account on Learning Log ❺ . This is the same link as in the header, but the button stands out on the page and shows the viewer what they need to do in order to start using the project. The selectors you see here style this as a large button that represents a call to action. The code `»` is an *HTML entity* that looks like two right angle brackets combined (`>>`). Finally, we provide closing `div` tags and close the `page_header` block. With only two `div` elements in this file, it's not particularly helpful to label the closing `div` tags. We aren't adding anything else to this page, so we don't need to define the `content` block in this template.

La página de inicio tiene ahora el aspecto de [la Figura 20-1](#) . ¡Se trata de una mejora significativa con respecto a la versión sin estilo del proyecto!

The home page now looks like [Figure 20-1](#) . This is a significant improvement over the unstyled version of the project!

Estilizar la página de inicio de sesión (Styling the Login Page)

Hemos refinado el aspecto general de la página de inicio de sesión, pero el propio formulario de inicio de sesión aún no tiene ningún estilo. Hagamos que el formulario tenga un aspecto coherente con el resto de la página modificando *login.xhtml* :

We've refined the overall appearance of the login page, but the login form itself doesn't have any styling yet. Let's make the form look consistent with the rest of the page by modifying *login.xhtml* :

login.xhtml

login.xhtml

```
{% extends 'learning_logs/base.xhtml' %}
❶ {% load django_bootstrap5 %}

❷ {% block page_header %}
    <h2>Log in to your account.</h2>
{% endblock page_header %}

{% block content %}

    <form action="{% url 'accounts:login' %}" method='post'>
        {% csrf_token %}
        ❸ {% bootstrap_form form %}
        ❹ {% bootstrap_button button_type="submit" content="Log in" %}
    </form>

{% endblock content %}
```

Primero cargamos las etiquetas de la plantilla `bootstrap5` en esta plantilla ❶ . A continuación, definimos el bloque `page_header` , que indica al usuario para qué página es ❷ . Observa que hemos eliminado el bloque `{% if form.errors %}` de la plantilla; `django-bootstrap5` gestiona los errores de formulario automáticamente.

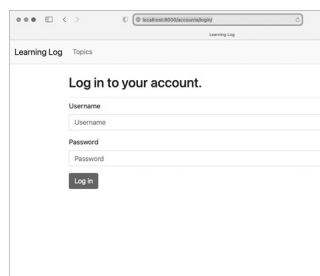
We first load the `bootstrap5` template tags into this template ❶ . We then define the `page_header` block, which tells the user what the page is for ❷ . Notice that we've removed the `{% if form.errors %}` block from the template; `django-bootstrap5` manages form errors automatically.

Para mostrar el formulario, utilizamos la etiqueta de plantilla `{% bootstrap_form %}` ❸ ; ésta sustituye al elemento `{{ form.as_div }}` que utilizábamos en el Capítulo 19 . La etiqueta de plantilla `{% bootstrap_form %}` inserta reglas de estilo Bootstrap en los elementos individuales del formulario a medida que éste se muestra. Para generar el botón de envío, utilizamos la etiqueta `{% bootstrap_button %}` con argumentos que lo designan como botón de envío, y le damos la etiqueta `Log in` ❹ .

To display the form, we use the template tag `{% bootstrap_form %}` ❸ ; this replaces the `{{ form.as_div }}` element we were using in Chapter 19 . The `{% bootstrap_form %}` template tag inserts Bootstrap style rules into the form's individual elements as the form is rendered. To generate the submit button, we use the `{% bootstrap_button %}` tag with arguments that designate it as a submit button, and give it the label `Log in` ❹ .

La Figura 20-2 muestra ahora el formulario de acceso. La página está mucho más limpia, con un estilo coherente y un propósito claro. Prueba a iniciar sesión con un nombre de usuario o una contraseña incorrectos; verás que incluso los mensajes de error tienen un estilo coherente y se integran bien en el conjunto del sitio.

Figure 20-2 shows the login form now. The page is much cleaner, with consistent styling and a clear purpose. Try logging in with an incorrect username or password; you'll see that even the error messages are styled consistently and integrate well with the overall site.



DESCRIPCIÓN DE LA
IMAGEN (En español):

Inicia sesión en tu
cuenta. Nombre de
usuario Contraseña
Iniciar sesión Temas
Registrar Iniciar sesión
Registro de
aprendizaje

Figura 20-2 : La página de inicio de sesión estilizada con Bootstrap

Figure 20-2 : The login page styled with Bootstrap

Estilizar la página de temas (Styling the Topics Page)

Asegurémonos de que las páginas para ver la información también tienen el estilo adecuado, empezando por la página de temas:

Let's make sure the pages for viewing information are styled appropriately as well, starting with the topics page:

topics.xhtml

topics.xhtml

```
{% extends 'learning_logs/base.xhtml' %}

{% block page_header %}
❶ <h1>Topics</h1>
{% endblock page_header %}

{% block content %}

❷ <ul class="list-group border-bottom pb-2 mb-4">
  {% for topic in topics %}
❸ <li class="list-group-item border-0">
    <a href="{% url 'learning_logs:topic' topic.id %}">
      {{ topic.text }}</a>
    </li>
    {% empty %}
❹ <li class="list-group-item border-0">No topics have been added yet.</li>
  {% endfor %}
</ul>

<a href="{% url 'learning_logs:new_topic' %}">Add a new topic</a>

{% endblock content %}
```

No necesitamos la etiqueta `{% load bootstrap5 %}`, porque no estamos utilizando ninguna etiqueta de plantilla personalizada de bootstrap5 en este archivo. Movemos el encabezamiento `Topics` al bloque `page_header` y lo convertimos en un elemento `<h1>` en lugar de un simple párrafo ❶.

We don't need the `{% load bootstrap5 %}` tag, because we're not using any custom bootstrap5 template tags in this file. We move the heading `Topics` into the `page_header` block and make it an `<h1>` element instead of a simple paragraph ❶ .

El contenido principal de esta página es una lista de temas, así que utilizamos el componente *list group* de Bootstrap para representar la página. Esto aplica un sencillo conjunto de directivas de estilo a la lista general y a cada elemento de la lista. Cuando abrimos la etiqueta `<ul>` , incluimos primero la clase `list-group` para aplicar las directivas de estilo por defecto a la lista ❷ . Personalizamos aún más la lista poniendo un borde en la parte inferior de la lista, un poco de relleno debajo de la lista (`pb-2`) y un margen debajo del borde inferior (`mb-4`).

The main content on this page is a list of topics, so we use Bootstrap's *list group* component to render the page. This applies a simple set of styling directives to the overall list and to each item in the list. When we open the `<ul>` tag, we first include the `list-group` class to apply the default style directives to the list ❷ . We further customize the list by putting a border at the bottom of the list, a little padding below the list (`pb-2`), and a margin below the bottom border (`mb-4`).

Cada elemento de la lista necesita la clase `list-group-item` , y personalizamos el estilo por defecto eliminando el borde alrededor de los elementos individuales ❸ . El mensaje que se muestra cuando la lista está vacía necesita estas mismas clases ❹ .

Each item in the list needs the `list-group-item` class, and we customize the default style by removing the border around individual items ❸ . The message that's displayed when the list is empty needs these same classes ❹ .

Ahora, cuando visites la página de temas, deberías ver una página con un estilo similar al de la página de inicio.

When you visit the topics page now, you should see a page with styling that matches the home page.

Estilizar las entradas de la página de temas (Styling the Entries on the Topic Page)

En la página de temas, utilizaremos el componente de tarjeta de Bootstrap para hacer que cada entrada destaque. Un *card* es un conjunto anidable de divs con estilos flexibles y predefinidos que son perfectos para mostrar las entradas de un tema:

On the topic page, we'll use Bootstrap's card component to make each entry stand out. A *card* is a nestable set of divs with flexible, predefined styles that are perfect for displaying a topic's entries:

topic.xhtml

topic.xhtml

```
{% extends 'learning_logs/base.xhtml' %}

❶ {% block page_header %}
    <h1>{{ topic.text }}</h1>
{% endblock page_header %}

{% block content %}
    <p>
        <a href="{% url 'learning_logs:new_entry' topic.id %}">Add new entry</a>
    </p>

    {% for entry in entries %}
❷    <div class="card mb-3">
        <!-- Card header with timestamp and edit link -->
❸    <h4 class="card-header">
        {{ entry.date_added|date:'M d, Y H:i' }}
❹    <small><a href="{% url 'learning_logs:edit_entry' entry.id %}">
        edit entry</a></small>
        </h4>
        <!-- Card body with entry text -->
❺    <div class="card-body">{{ entry.text|linebreaks }}</div>
        </div>
    {% empty %}
❻    <p>There are no entries for this topic yet.</p>
```

```
{% endfor %}
```

```
{% endblock content %}
```

Primero colocamos el tema en el bloque `page_header` ❶ . A continuación, eliminamos la estructura de lista desordenada utilizada anteriormente en esta plantilla. En lugar de hacer de cada entrada un elemento de lista, abrimos un elemento `div` con el selector `card` ❷ . Esta tarjeta tiene dos elementos anidados: uno para contener la marca de tiempo y el enlace para editar la entrada, y otro para contener el cuerpo de la entrada. El selector `card` se encarga de la mayor parte del estilo que necesitamos para este `div`; personalizamos la tarjeta añadiendo un pequeño margen en la parte inferior de cada tarjeta (`mb-3`).

We first place the topic in the `page_header` block ❶ . Then we delete the unordered list structure previously used in this template. Instead of making each entry a list item, we open a `div` element with the selector `card` ❷ . This card has two nested elements: one to hold the timestamp and the link to edit the entry, and another to hold the body of the entry. The `card` selector takes care of most of the styling we need for this `div`; we customize the card by adding a small margin to the bottom of each card (`mb-3`).

El primer elemento de la tarjeta es una cabecera, que es un elemento `<h4>` con el selector `card-header` ❸ . Esta cabecera contiene la fecha en que se hizo la entrada y un enlace para editarla. La etiqueta `<small>` alrededor del enlace `edit_entry` hace que aparezca un poco más pequeño que la marca de tiempo ❹ . El segundo elemento es un `div` con el selector `card-body` ❺ , que coloca el texto de la entrada en una simple caja en la tarjeta. Observa que el código Django para incluir la información en la página no ha cambiado; sólo lo han hecho los elementos que afectan a la apariencia de la página. Como ya no tenemos una lista desordenada, hemos sustituido las etiquetas de elemento de lista alrededor del mensaje de lista vacía por simples etiquetas de párrafo ❻ .

The first element in the card is a header, which is an `<h4>` element with the selector `card-header` ❸ . This header contains the date the entry was made and a link to edit the entry. The `<small>` tag around the `edit_entry` link makes it appear a little smaller than the timestamp ❹ . The second element is a `div` with the selector `card-body` ❺ , which places the text of the entry in a simple box on the card. Notice that the Django code for including the information on the page hasn't changed; only elements that affect the appearance of the page have. Since we no longer have an unordered list, we've replaced the list item tags around the empty list message with simple paragraph tags ❻ .

La Figura 20-3 muestra la página de temas con su nuevo aspecto. La funcionalidad de Learning Log no ha cambiado, pero tiene un aspecto mucho más profesional y atractivo para los usuarios.

Figure 20-3 shows the topic page with its new look. Learning Log's functionality hasn't changed, but it looks significantly more professional and inviting to users.

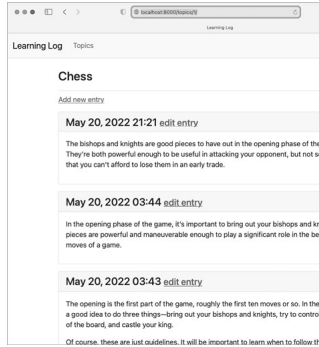
Si quieres utilizar una plantilla Bootstrap diferente para un proyecto, sigue un proceso similar al que hemos hecho hasta ahora en este capítulo. Copia la plantilla que quieras utilizar en *base.xhtml* , y modifica los elementos que contienen contenido real para que la plantilla muestre la información de tu proyecto. A continuación, utiliza las herramientas de estilo individuales de Bootstrap para dar estilo al contenido de cada página.

If you want to use a different Bootstrap template for a project, follow a process that's similar to what we've done so far in this chapter. Copy the template you want to use into *base.xhtml* , and modify the elements that contain actual content so the template displays your project's information. Then use Bootstrap's individual styling tools to style the content on each page.

Nota (Note)

El proyecto Bootstrap tiene una documentación excelente. Visita la página principal en <https://getbootstrap.com> y haz clic en **Docs** para saber más sobre lo que ofrece Bootstrap.

The Bootstrap project has excellent documentation. Visit the home page at <https://getbootstrap.com> and click **Docs** to learn more about what Bootstrap offers.



DESCRIPCIÓN DE LA IMAGEN (En español):

Registro de Aprendizaje Ajedrez
Agregar nueva entrada
20 de mayo de 2022 21:21 editar entrada
Los alfiles y caballos son buenas piezas para tener en la fase de apertura del juego. Son ambos útiles para atacar a tu oponente, pero no tan poderosos como para querer perderlos en un intercambio fácil. 20 de mayo de 2022 03:44 editar entrada En la fase de apertura del juego, es importante sacar tus alfiles y caballos. Estas piezas son poderosas y manejarlas temprano puede desempeñar un papel significativo al principio de una partida de ajedrez. 20 de mayo de 2022 03:23 editar entrada La apertura es la primera parte del juego, aproximadamente los primeros diez movimientos o así. En la apertura, es buena idea hacer tres cosas: sacar tus alfiles y caballos, intentar controlar el centro del tablero e enrocarte.

Figura 20-3 : La página del tema con el estilo Bootstrap

Figure 20-3 : The topic page with Bootstrap styling

Pruébalo tú mismo *(Try It Yourself)*

20-1. Otros formularios: Hemos aplicado los estilos de Bootstrap a la página login . Haz cambios similares en el resto de las páginas basadas en formularios, incluyendo new_topic , new_entry , edit_entry , y register .

20-1. Other Forms: We applied Bootstrap's styles to the login page. Make similar changes to the rest of the form-based pages, including new_topic , new_entry , edit_entry , and register .

20-2. Blog con estilo: Utiliza Bootstrap para dar estilo al proyecto Blog que creaste en el Capítulo 19 .

20-2. Stylish Blog: Use Bootstrap to style the Blog project you created in Chapter 19 .

Despliegue del Registro de Aprendizaje *(Deploying Learning Log)*

Ahora que tenemos un proyecto de aspecto profesional, vamos a desplegarlo en un servidor en vivo para que cualquier persona con conexión a Internet pueda utilizarlo. Utilizaremos Platform.sh, una plataforma basada en web que te permite gestionar el despliegue de aplicaciones web. Pondremos en marcha Learning Log en Platform.sh.

Now that we have a professional-looking project, let's deploy it to a live server so anyone with an internet connection can use it. We'll use Platform.sh, a web-based platform that allows you to manage the deployment of web applications. We'll get Learning Log up and running on Platform.sh.

Crear una cuenta en Platform.sh *(Making a Platform.sh Account)*

Para crear una cuenta, ve a <https://platform.sh> y haz clic en el botón **Prueba gratuita** . Platform.sh tiene un nivel gratuito que, en el momento de escribir esto, no requiere tarjeta de crédito. El periodo de prueba te permite desplegar una aplicación con recursos mínimos, lo que te permite probar tu proyecto en un despliegue en vivo antes de comprometerte con un plan de alojamiento de pago.

To make an account, go to <https://platform.sh> and click the **Free Trial** button. Platform.sh has a free tier that, as of this writing, does not require a credit card. The trial period allows you to deploy an app with minimal resources, which lets you test your project in a live deployment before committing to a paid hosting plan.

Nota (Note)

Los límites específicos de los planes de prueba suelen cambiar periódicamente, ya que las plataformas de alojamiento luchan contra el spam y el abuso de recursos. Puedes ver los límites actuales de la prueba gratuita en <https://platform.sh/free-trial>.

The specific limits of trial plans tend to change periodically, as hosting platforms fight spam and abuse of resources. You can see the current limits of the free trial at <https://platform.sh/free-trial>.

Instalación de la CLI Platform.sh (Installing the Platform.sh CLI)

Para desplegar y gestionar un proyecto en Platform.sh, necesitarás las herramientas disponibles en la interfaz de línea de comandos (CLI). Para instalar la última versión de la CLI, visita <https://docs.platform.sh/development/cli.xhtml> y sigue las instrucciones correspondientes a tu sistema operativo.

To deploy and manage a project on Platform.sh, you'll need the tools available in the Command Line Interface (CLI). To install the latest version of the CLI, visit <https://docs.platform.sh/development/cli.xhtml> and follow the instructions for your operating system.

En la mayoría de los sistemas, puedes instalar la CLI ejecutando el siguiente comando en un terminal:

On most systems, you can install the CLI by running the following command in a terminal:

```
$ curl -fsS https://platform.sh/cli/installer | php
```

Una vez finalizada la ejecución de este comando, deberás abrir una nueva ventana de terminal antes de poder utilizar la CLI.

After this command has finished running, you will need to open a new terminal window before you can use the CLI.

Nota (Note)

Es probable que este comando no funcione en un terminal estándar en Windows. Puedes utilizar el Subsistema de Windows para Linux (WSL) o un terminal Git Bash. Si necesitas instalar PHP, puedes utilizar el instalador de XAMPP de <https://apachefriends.org> . Si tienes alguna dificultad para instalar la CLI Platform.sh, consulta las instrucciones de instalación más detalladas del Apéndice E .

This command will probably not work in a standard terminal on Windows. You can use Windows Subsystem for Linux (WSL) or a Git Bash terminal. If you need to install PHP, you can use the XAMPP installer from <https://apachefriends.org> . If you have any difficulty installing the Platform.sh CLI, see the more detailed installation instructions in Appendix E .

Instalación de platformshconfig (Installing platformshconfig)

También necesitarás instalar un paquete adicional, `platformshconfig` . Este paquete ayuda a detectar si el proyecto se está ejecutando en tu sistema local o en un servidor Platform.sh. En un entorno virtual activo, emite el siguiente comando:

You'll also need to install one additional package, `platformshconfig` . This package helps detect whether the project is running on your local system or on a Platform.sh server. In an active virtual environment, issue the following command:

```
(ll_env)learning_log$ pip install platformshconfig
```

Utilizaremos este paquete para modificar la configuración del proyecto cuando se esté ejecutando en el servidor activo.

We'll use this package to modify the project's settings when it's running on the live server.

Crear un archivo `requirements.txt` (*Creating a requirements.txt File*)

El servidor remoto necesita saber de qué paquetes depende Learning Log, así que utilizaremos pip para generar un archivo que los enumere. De nuevo, desde un entorno virtual activo, emite el siguiente comando:

The remote server needs to know which packages Learning Log depends on, so we'll use pip to generate a file listing them. Again, from an active virtual environment, issue the following command:

```
(ll_env)learning_log$ pip freeze > requirements.txt
```

El comando `freeze` indica a pip que escriba los nombres de todos los paquetes instalados actualmente en el proyecto en el archivo `requirements.txt`. Abre este archivo para ver los paquetes y números de versión instalados en tu proyecto:

The `freeze` command tells pip to write the names of all the packages currently installed in the project into the file `requirements.txt`. Open this file to see the packages and version numbers installed in your project:

requisitos.txt

requirements.txt

```
asgiref==3.5.2
beautifulsoup4==4.11.1
Django==4.1
django-bootstrap5==21.3
platformshconfig==2.4.0
soupsieve==2.3.2.post1
sqlparse==0.4.2
```

Learning Log ya depende de versiones específicas de siete paquetes diferentes, por lo que requiere un entorno adecuado para ejecutarse correctamente en un servidor remoto. (Instalamos tres de estos paquetes manualmente, y cuatro de ellos se instalaron automáticamente como dependencias de estos paquetes).

Learning Log already depends on specific versions of seven different packages, so it requires a matching environment to run properly on a remote server. (We installed three of these packages manually, and four of them were installed automatically as dependencies of these packages.)

Cuando despluguemos Learning Log, Platform.sh instalará todos los paquetes enumerados en *requirements.txt*, creando un entorno con los mismos paquetes que estamos utilizando localmente. Por ello, podemos estar seguros de que el proyecto desplegado funcionará igual que en nuestro sistema local. Este enfoque de la gestión de un proyecto es fundamental cuando empiezas a crear y mantener varios proyectos en tu sistema.

When we deploy Learning Log, Platform.sh will install all the packages listed in *requirements.txt*, creating an environment with the same packages we're using locally. Because of this, we can be confident the deployed project will function just like it has on our local system. This approach to managing a project is critical as you start to build and maintain multiple projects on your system.

Nota (Note)

Si el número de versión de un paquete que aparece en tu sistema difiere del que se muestra aquí, quédate con la versión que tienes en tu sistema.

If the version number for a package listed on your system differs from what's shown here, keep the version you have on your system.

Requisitos adicionales para el despliegue (Additional Deployment Requirements)

El servidor en vivo requiere dos paquetes adicionales. Estos paquetes se utilizan para servir el proyecto en un entorno de producción, donde muchos usuarios pueden estar haciendo peticiones al mismo tiempo.

The live server requires two additional packages. These packages are used to serve the project in a production environment, where many users can be making requests at the same time.

En el mismo directorio donde está guardado *requirements.txt* , crea un nuevo archivo llamado *requirements_remote.txt* . Añádele los dos paquetes siguientes:

In the same directory where *requirements.txt* is saved, make a new file called *requirements_remote.txt* . Add the following two packages to it:

requisitos_remoto.txt

requirements_remote.txt

```
# Requirements for live project.  
gunicorn  
psycogp2
```

El paquete `gunicorn` responde a las peticiones a medida que llegan al servidor remoto; éste ocupa el lugar del servidor de desarrollo que hemos estado utilizando localmente. El paquete `psycogp2` es necesario para que Django gestione la base de datos Postgres que utiliza Platform.sh. *Postgres* es una base de datos de código abierto muy adecuada para aplicaciones de producción.

The `gunicorn` package responds to requests as they come in to the remote server; this takes the place of the development server we've been using locally. The `psycopg2` package is required to let Django manage the Postgres database that Platform.sh uses. *Postgres* is an open source database that's extremely well suited to production apps.

Añadir archivos de configuración *(Adding Configuration Files)*

Cada plataforma de alojamiento requiere alguna configuración para que un proyecto se ejecute correctamente en sus servidores. En esta sección, añadiremos tres archivos de configuración:

Every hosting platform requires some configuration for a project to run correctly on its servers. In this section, we'll add three configuration files:

.platform.app.yaml Este es el archivo de configuración principal del proyecto. Indica a Platform.sh qué tipo de proyecto estamos intentando desplegar y qué tipo de recursos necesita nuestro proyecto, e incluye comandos para construir el proyecto en el servidor.

.platform.app.yaml This is the main configuration file for the project. This tells Platform.sh what kind of project we're trying to deploy and what kinds of resources our project needs, and it includes commands for building the project on the server.

.platform/routes.yaml Este archivo define las rutas a nuestro proyecto. Cuando Platform.sh recibe una solicitud, ésta es la configuración que ayuda a dirigir estas solicitudes a nuestro proyecto específico.

.platform/routes.yaml This file defines the routes to our project. When a request is received by Platform.sh, this is the configuration that helps direct these requests to our specific project.

.platform/services.yaml Este archivo define cualquier servicio adicional que necesite nuestro proyecto.

.platform/services.yaml This file defines any additional services our project needs.

Todos estos son archivos YAML (YAML Ain't Markup Language). *YAML* es un lenguaje diseñado para escribir archivos de configuración; está hecho para ser leído fácilmente tanto por humanos como por ordenadores. Puedes escribir o modificar a mano un archivo YAML típico, pero un ordenador también puede leerlo e interpretarlo sin ambigüedades.

These are all YAML (YAML Ain't Markup Language) files. *YAML* is a language designed for writing configuration files; it's made to be read easily by both humans and computers. You can write or modify a typical YAML file by hand, but a computer can also read and interpret the file unambiguously.

Los archivos YAML son magníficos para la configuración del despliegue, porque te dan un gran control sobre lo que ocurre durante el proceso de despliegue.

YAML files are great for deployment configuration, because they give you a good deal of control over what happens during the deployment process.

Hacer visibles los archivos ocultos (Making Hidden Files Visible)

La mayoría de los sistemas operativos ocultan los archivos y carpetas que empiezan por un punto, como *.platform* . Cuando abras un explorador de archivos, no verás este tipo de archivos y carpetas por defecto. Pero como programador, necesitarás verlos. He aquí cómo ver los archivos ocultos, dependiendo de tu sistema operativo:

Most operating systems hide files and folders that begin with a dot, such as *.platform* . When you open a file browser, you won't see these kinds of files and folders by default. But as a programmer, you'll need to see them. Here's how to view hidden files, depending on your operating system:

Listado en español:

- En Windows, abre el Explorador de Windows y, a continuación, abre una carpeta como *Desktop* . Haz clic en la pestaña **Ver** y asegúrate de que están marcadas las opciones **Extensiones de nombre de archivo** y **Elementos ocultos** .
- En macOS, puedes pulsar **⌘**-SHIFT-. (punto) en cualquier ventana del Finder para ver los archivos y carpetas ocultos.

- En sistemas Linux como Ubuntu, puedes pulsar CTRL-H en cualquier explorador de archivos para ver los archivos y carpetas ocultos. Para que esta configuración sea permanente, abre un explorador de archivos como Nautilus y haz clic en la pestaña de opciones (indicada por tres líneas). Selecciona la casilla **Mostrar archivos ocultos**.

Listing in English:

- On Windows, open Windows Explorer, and then open a folder such as *Desktop*. Click the **View** tab, and make sure **File name extensions** and **Hidden items** are checked.
- On macOS, you can press ⌘-SHIFT-. (dot) in any Finder window to see hidden files and folders.
- On Linux systems such as Ubuntu, you can press CTRL-H in any file browser to display hidden files and folders. To make this setting permanent, open a file browser such as Nautilus and click the options tab (indicated by three lines). Select the **Show Hidden Files** checkbox.

El archivo de configuración .platform.app.yaml (The .platform.app.yaml Configuration File)

El primer archivo de configuración es el más largo, porque controla el proceso general de despliegue. Lo mostraremos por partes; puedes introducirlo a mano en tu editor de texto o descargar una copia de los recursos en línea en https://ehmatthes.github.io/pcc_3e.

The first configuration file is the longest, because it controls the overall deployment process. We'll show it in parts; you can either enter it by hand in your text editor or download a copy from the online resources at https://ehmatthes.github.io/pcc_3e.

Aquí tienes la primera parte de *.platform.app.yaml*, que debes guardar en el mismo directorio que *manage.py*:

Here's the first part of *.platform.app.yaml*, which should be saved in the same directory as *manage.py*:

.platform.app.yaml

.platform.app.yaml

```
❶ name: "ll_project"
type: "python:3.10"

❷ relationships:
  database: "db:postgresql"

# The configuration of the app when it's exposed to the web.
❸ web:
  upstream:
    socket_family: unix
  commands:
❹   start: "gunicorn -w 4 -b unix:$SOCKET ll_project.wsgi:application"
❺  locations:
    "/":
      passthru: true
    "/static":
      root: "static"
      expires: 1h
      allow: true

# The size of the persistent disk of the application (in MB).
❻ disk: 512
```

Cuando guardes este archivo, asegúrate de incluir el punto al principio del nombre del archivo. Si omites el punto, Platform.sh no encontrará el archivo y tu proyecto no se desplegará.

When you save this file, make sure you include the dot at the beginning of the filename. If you omit the dot, Platform.sh won't find the file and your project will not be deployed.

No es necesario que lo entiendas todo en *.platform.app.yaml* en este momento; destacaré las partes más importantes de la configuración. El archivo comienza especificando el `name` del proyecto, que llamaremos `'ll_project'` para ser coherentes con el nombre que utilizamos al iniciar el proyecto ❶. También tenemos que especificar la versión de Python que estamos utilizando (3.10 en el momento de escribir esto). Puedes encontrar una lista de las versiones compatibles en <https://docs.platform.sh/languages/python.xhtml>.

You don't need to understand everything in *.platform.app.yaml* at this point; I'll highlight the most important parts of the configuration. The file starts off by specifying the `name` of the project, which we're calling `'ll_project'` to be consistent with the name we used when

starting the project ❶ . We also need to specify the version of Python we're using (3.10 at the time of this writing). You can find a list of supported versions at <https://docs.platform.sh/languages/python.xhtml> .

A continuación hay una sección etiquetada `relationships` que define otros servicios que el proyecto necesita ❷ . Aquí la única relación es con una base de datos Postgres. Después está la sección `web` ❸ . La sección `commands:start` indica a Platform.sh qué proceso debe utilizar para atender las peticiones entrantes. Aquí estamos especificando que `gunicorn` gestionará las peticiones ❹ . Este comando sustituye al comando `python manage.py runserver` que hemos estado utilizando localmente.

Next is a section labeled `relationships` that defines other services the project needs ❷ . Here the only relationship is to a Postgres database. After that is the `web` section ❸ . The `commands:start` section tells Platform.sh what process to use to serve incoming requests. Here we're specifying that `gunicorn` will handle requests ❹ . This command takes the place of the `python manage.py runserver` command we've been using locally.

La sección `locations` indica a Platform.sh dónde enviar las solicitudes entrantes ❺ . La mayoría de las solicitudes deben enviarse a `gunicorn` ; nuestros archivos `urls.py` indicarán a `gunicorn` exactamente cómo gestionarlas. Las peticiones de archivos estáticos se gestionarán por separado y se actualizarán una vez cada hora. La última línea muestra que estamos solicitando 512 MB de espacio en disco en uno de los servidores de Platform.sh ❻ .

The `locations` section tells Platform.sh where to send incoming requests ❺ . Most requests should be passed through to `gunicorn` ; our `urls.py` files will tell `gunicorn` exactly how to handle those requests. Requests for static files will be handled separately and will be refreshed once an hour. The last line shows that we're requesting 512MB of disk space on one of Platform.sh's servers ❻ .

El resto de [.platform.app.yaml](#) es como sigue:

The rest of [.platform.app.yaml](#) is as follows:

```
--snip--
disk: 512
```

```
# Set a local read/write mount for logs.
```

```
❶ mounts:
```

```
  "logs":
```

```
    source: local
```

```
    source_path: logs
```

```
# The hooks executed at various points in the lifecycle of the application.
```

```
❷ hooks:
```

```
  build: |
```

```
❸    pip install --upgrade pip
```

```
    pip install -r requirements.txt
```

```
    pip install -r requirements_remote.txt
```

```
    mkdir logs
```

```
❹    python manage.py collectstatic
```

```
    rm -rf logs
```

```
❺  deploy: |
```

```
    python manage.py migrate
```

La sección `mounts` ❶ nos permite definir directorios en los que podemos leer y escribir datos mientras se ejecuta el proyecto. Esta sección define un directorio `logs/` para el proyecto desplegado.

The `mounts` section ❶ lets us define directories where we can read and write data while the project is running. This section defines a `logs/` directory for the deployed project.

La sección `hooks` ❷ define las acciones que se llevan a cabo en varios puntos durante el proceso de despliegue. En la sección `build`, instalamos todos los paquetes necesarios para servir el proyecto en el entorno en vivo ❸. También ejecutamos `collectstatic` ❹, que recopila todos los archivos estáticos necesarios para el proyecto en un solo lugar para que puedan ser servidos eficientemente.

The `hooks` section ❷ defines actions that are taken at various points during the deployment process. In the `build` section, we install all the packages that are required to serve the project in the live environment ❸. We also run `collectstatic` ❹, which collects all the static files needed for the project into one place so they can be served efficiently.

Por último, en la sección `deploy` ⑤ , especificamos que las migraciones deben ejecutarse cada vez que se despliegue el proyecto. En un proyecto sencillo, esto no tendrá ningún efecto cuando no haya habido cambios.

Finally, in the `deploy` section ⑤ , we specify that migrations should be run each time the project is deployed. In a simple project, this will have no effect when there have been no changes.

Los otros dos archivos de configuración son mucho más cortos; vamos a escribirlos ahora.

The other two configuration files are much shorter; let's write them now.

El archivo de configuración `routes.yaml` (The `routes.yaml` Configuration File)

Un *route* es el camino que sigue una petición cuando es procesada por el servidor. Cuando Platform.sh recibe una solicitud, necesita saber dónde enviarla.

A *route* is the path a request takes as it's processed by the server. When a request is received by Platform.sh, it needs to know where to send the request.

Crea una nueva carpeta llamada `.platform` , en el mismo directorio que `manage.py` . Asegúrate de incluir el punto al principio del nombre. Dentro de esa carpeta, crea un archivo llamado `routes.yaml` e introduce lo siguiente:

Make a new folder called `.platform` , in the same directory as `manage.py` . Make sure you include the dot at the beginning of the name. Inside that folder, make a file called `routes.yaml` and enter the following:

`.platform/rutas.yaml`

.platform/routes.yaml

Each route describes how an incoming URL will be processed by Platform.sh.

```
"https://{default}":  
  type: upstream  
  upstream: "ll_project:http"
```

```
"https://www.{default}":  
  type: redirect  
  to: "https://{default}"
```

Este archivo garantiza que las peticiones como *https://project_url.com* y *www.project_url.com* se dirijan al mismo lugar.

This file makes sure requests like *https://project_url.com* and *www.project_url.com* all get routed to the same place.

El archivo de configuración services.yaml (The services.yaml Configuration File)

Este último archivo de configuración especifica los servicios que nuestro proyecto necesita para funcionar. Guarda este archivo en el directorio *.platform/*, junto a *routes.yaml* :

This last configuration file specifies services that our project needs in order to run. Save this file in the *.platform/* directory, alongside *routes.yaml* :

.platform/routes.yaml

Each service listed will be deployed in its own container as part of your
Platform.sh project.

db:

```
type: postgresql:12
disk: 1024
```

Este archivo define un servicio, una base de datos Postgres.

This file defines one service, a Postgres database.

Modificar *settings.py* para Platform.sh (Modifying *settings.py* for Platform.sh)

Ahora tenemos que añadir una sección al final de *settings.py* para modificar algunos ajustes para el entorno Platform.sh. Añade este código al final de *settings.py* :

Now we need to add a section at the end of *settings.py* to modify some settings for the Platform.sh environment. Add this code to the very end of *settings.py* :

settings.py

settings.py

```
--snip--
# Platform.sh settings.
❶ from platformshconfig import Config

config = Config()
❷ if config.is_valid_platform():
❸     ALLOWED_HOSTS.append('.platformsh.site')

❹ if config.appDir:
    STATIC_ROOT = Path(config.appDir) / 'static'
❺ if config.projectEntropy:
    SECRET_KEY = config.projectEntropy

if not config.in_build():
❻     db_settings = config.credentials('database')
    DATABASES = {
        'default': {
```

```

        'ENGINE': 'django.db.backends.postgresql',
        'NAME': db_settings['path'],
        'USER': db_settings['username'],
        'PASSWORD': db_settings['password'],
        'HOST': db_settings['host'],
        'PORT': db_settings['port'],
    },
}

```

Normalmente colocamos las declaraciones `import` al principio de un módulo, pero en este caso, es útil mantener todas las configuraciones específicas remotas en una sola sección. Aquí importamos `Config` de `platformshconfig` ❶, que ayuda a determinar la configuración en el servidor remoto. Sólo modificamos la configuración si el método `config.is_valid_platform()` devuelve `True` ❷, lo que indica que la configuración se está utilizando en un servidor `Platform.sh`.

We normally place `import` statements at the beginning of a module, but in this case, it's helpful to keep all the remote-specific settings in one section. Here we import `Config` from `platformshconfig` ❶, which helps determine settings on the remote server. We only modify settings if the method `config.is_valid_platform()` returns `True` ❷, indicating the settings are being used on a `Platform.sh` server.

Modificamos `ALLOWED_HOSTS` para permitir que el proyecto sea servido por hosts que terminen en `.platformsh.site` ❸. Todos los proyectos desplegados en la capa gratuita se servirán utilizando este host. Si la configuración se carga en el directorio de la aplicación desplegada ❹, configuramos `STATIC_ROOT` para que los archivos estáticos se sirvan correctamente. También establecemos un `SECRET_KEY` más seguro en el servidor remoto ❺.

We modify `ALLOWED_HOSTS` to allow the project to be served by hosts ending in `.platformsh.site` ❸. All projects deployed to the free tier will be served using this host. If settings are being loaded in the deployed app's directory ❹, we set `STATIC_ROOT` so that static files are served correctly. We also set a more secure `SECRET_KEY` on the remote server ❺.

Por último, configuramos la base de datos de producción ❻. Esto sólo se configura si el proceso de compilación ha terminado de ejecutarse y se está sirviendo el proyecto. Todo lo que ves aquí es necesario para permitir que Django hable con el servidor Postgres que

Platform.sh configuró para el proyecto.

Finally, we configure the production database ⑥ . This is only set if the build process has finished running and the project is being served. Everything you see here is necessary to let Django talk to the Postgres server that Platform.sh set up for the project.

Utilizar Git para rastrear los archivos del proyecto (Using Git to Track the Project's Files)

Como se comentó en el Capítulo 17 , Git es un programa de control de versiones que te permite tomar una instantánea del código de tu proyecto cada vez que implementas con éxito una nueva característica. Si algo va mal, puedes volver fácilmente a la última instantánea funcional de tu proyecto; por ejemplo, si accidentalmente introduces un error mientras trabajas en una nueva función. Cada instantánea se denomina *commit* .

As discussed in Chapter 17 , Git is a version control program that allows you to take a snapshot of the code in your project each time you implement a new feature successfully. If anything goes wrong, you can easily return to the last working snapshot of your project; for example, if you accidentally introduce a bug while working on a new feature. Each snapshot is called a *commit* .

Utilizando Git, puedes probar a implementar nuevas funciones sin preocuparte de romper tu proyecto. Cuando despliegues en un servidor activo, debes asegurarte de que estás desplegando una versión funcional de tu proyecto. Para leer más sobre Git y el control de versiones, consulta el Apéndice D .

Using Git, you can try implementing new features without worrying about breaking your project. When you're deploying to a live server, you need to make sure you're deploying a working version of your project. To read more about Git and version control, see Appendix D .

Instalar Git (Installing Git)

Puede que Git ya esté instalado en tu sistema. Para averiguarlo, abre una nueva ventana de terminal y emite el comando `git --version` :

Git may already be installed on your system. To find out, open a new terminal window and issue the command `git --version` :

```
(ll_env)learning_log$ git --version  
git version 2.30.1 (Apple Git-130)
```

Si aparece un mensaje indicando que Git no está instalado, consulta las instrucciones de instalación en el Apéndice D .

If you get a message indicating that Git is not installed, see the installation instructions in Appendix D .

Configurar Git (Configuring Git)

Git realiza un seguimiento de quién realiza cambios en un proyecto, incluso cuando sólo una persona está trabajando en el proyecto. Para ello, Git necesita saber tu nombre de usuario y tu correo electrónico. Debes proporcionar un nombre de usuario, pero puedes inventarte un correo electrónico para tus proyectos de prácticas:

Git keeps track of who makes changes to a project, even when only one person is working on the project. To do this, Git needs to know your username and email. You must provide a username, but you can make up an email for your practice projects:

```
(ll_env)learning_log$ git config --global user.name "eric"  
(ll_env)learning_log$ git config --global user.email "eric@example.com"
```

Si olvidas este paso, Git te pedirá esta información cuando hagas tu primera confirmación.

If you forget this step, Git will prompt you for this information when you make your first commit.

Ignorar archivos (Ignoring Files)

No necesitamos que Git rastree todos los archivos del proyecto, así que le diremos que ignore algunos archivos. Crea un archivo llamado `.gitignore` en la carpeta que contiene `manage.py` . Observa que este nombre de archivo comienza con un punto y no tiene

extensión. Este es el código que va en *.gitignore* :

We don't need Git to track every file in the project, so we'll tell it to ignore some files. Create a file called *.gitignore* in the folder that contains *manage.py* . Notice that this filename begins with a dot and has no file extension. Here's the code that goes in *.gitignore* :

.gitignore

.gitignore

```
ll_env/  
__pycache__/  
*.sqlite3
```

Le decimos a Git que ignore todo el directorio *ll_env* , porque podemos volver a crearlo automáticamente en cualquier momento. Tampoco rastreamos el directorio *__pycache__* , que contiene los archivos *.pyc* que se crean automáticamente cuando se ejecutan los archivos *.py* . No hacemos un seguimiento de los cambios en la base de datos local, porque es un mal hábito: si alguna vez utilizas SQLite en un servidor, podrías sobrescribir accidentalmente la base de datos activa con tu base de datos de prueba local cuando envíes el proyecto al servidor. El asterisco en **.sqlite3* indica a Git que ignore cualquier archivo que termine con la extensión *.sqlite3* .

We tell Git to ignore the entire *ll_env* directory, because we can re-create it automatically at any time. We also don't track the *__pycache__* directory, which contains the *.pyc* files that are created automatically when the *.py* files are executed. We don't track changes to the local database, because it's a bad habit: if you're ever using SQLite on a server, you might accidentally overwrite the live database with your local test database when you push the project to the server. The asterisk in **.sqlite3* tells Git to ignore any file that ends with the extension *.sqlite3* .

Nota (Note)

Si utilizas macOS, añade `.DS_Store` a tu archivo `.gitignore`. Se trata de un archivo que almacena información sobre la configuración de carpetas en macOS, y no tiene nada que ver con este proyecto.

If you're using macOS, add `.DS_Store` to your `.gitignore` file. This is a file that stores information about folder settings on macOS, and it has nothing to do with this project.

Confirmar el proyecto (Committing the Project)

Tenemos que inicializar un repositorio Git para Learning Log, añadir todos los archivos necesarios al repositorio y confirmar el estado inicial del proyecto. He aquí cómo hacerlo:

We need to initialize a Git repository for Learning Log, add all the necessary files to the repository, and commit the initial state of the project. Here's how to do that:

```
❶ (ll_env)learning_log$ git init
Initialized empty Git repository in /Users/eric/.../learning_log/.git/
❷ (ll_env)learning_log$ git add .
❸ (ll_env)learning_log$ git commit -am "Ready for deployment to Platform.sh."
[main (root-commit) c7ffaad] Ready for deployment to Platform.sh.
42 files changed, 879 insertions(+)
create mode 100644 .gitignore
create mode 100644 .platform.app.yaml
--snip--
create mode 100644 requirements_remote.txt
❹ (ll_env)learning_log$ git status
On branch main
nothing to commit, working tree clean
(ll_env)learning_log$
```

Ejecutamos el comando `git init` para inicializar un repositorio vacío en el directorio que contiene Learning Log ❶. A continuación, utilizamos el comando `git add .`, que añade todos los archivos que no se están ignorando al repositorio ❷. (No olvides el punto.) A continuación, emitimos el comando `git commit -am "commit message"`: la bandera `-a` indica a Git que incluya todos los archivos modificados en esta confirmación, y la bandera

`-m` indica a Git que registre un mensaje de registro ❸ .

We issue the `git init` command to initialize an empty repository in the directory containing Learning Log ❶ . We then use the `git add .` command, which adds all the files that aren't being ignored to the repository ❷ . (Don't forget the dot.) Next, we issue the command `git commit -am "commit message"` : the `-a` flag tells Git to include all changed files in this commit, and the `-m` flag tells Git to record a log message ❸ .

Emitir el comando `git status` ❹ indica que estamos en la rama *main* y que nuestro árbol de trabajo es *clean* . Este es el estado que querrás ver cada vez que envíes tu proyecto a un servidor remoto.

Issuing the `git status` command ❹ indicates that we're on the *main* branch and that our working tree is *clean* . This is the status you'll want to see anytime you push your project to a remote server.

Crear un proyecto en Plataforma.sh (*Creating a Project on Platform.sh*)

Llegados a este punto, el proyecto Learning Log sigue ejecutándose en nuestro sistema local y también está configurado para ejecutarse correctamente en un servidor remoto. Utilizaremos la CLI Plataforma.sh para crear un nuevo proyecto en el servidor y, a continuación, enviaremos nuestro proyecto al servidor remoto.

At this point, the Learning Log project still runs on our local system and is also configured to run correctly on a remote server. We'll use the Platform.sh CLI to create a new project on the server and then push our project to the remote server.

Asegúrate de que estás en un terminal, en el directorio *learning_log/* , y emite el siguiente comando:

Make sure you're in a terminal, at the *learning_log/* directory, and issue the following command:

```
(ll_env)learning_log$ platform login
Opened URL: http://127.0.0.1:5000
Please use the browser to log in.
```

--snip--

❶ Do you want to create an SSH configuration file automatically? [Y/n] Y

Este comando abrirá una pestaña del navegador donde podrás iniciar sesión. Una vez iniciada la sesión, puedes cerrar la pestaña del navegador y volver al terminal. Si se te pide que crees un archivo de configuración SSH ❶ , introduce Y para que puedas conectarte al servidor remoto más tarde.

This command will open a browser tab where you can log in. Once you're logged in, you can close the browser tab and return to the terminal. If you're prompted about creating an SSH configuration file ❶ , enter Y so you can connect to the remote server later.

Ahora crearemos un proyecto. Hay mucha salida, así que veremos el proceso de creación por secciones. Empieza emitiendo el comando create :

Now we'll create a project. There's a lot of output, so we'll look at the creation process in sections. Start by issuing the create command:

```
(ll_env)learning_log$ platform create
```

* Project title (--title)

Default: Untitled Project

❶ > ll_project

* Region (--region)

The region where the project will be hosted

--snip--

[us-3.platform.sh] Moses Lake, United States (AZURE) [514 gC02eq/kWh]

❷ > us-3.platform.sh

* Plan (--plan)

Default: development

Enter a number to choose:

[0] development

--snip--

❸ > 0

* Environments (--environments)

The number of environments

Default: 3

④ > 3

* Storage (--storage)

The amount of storage per environment, in GiB

Default: 5

⑤ > 5

El primer mensaje pide un nombre para el proyecto ① , así que utilizaremos el nombre `ll_project` . El siguiente mensaje pregunta en qué región queremos que esté el servidor ② . Elige el servidor que te quede más cerca; en mi caso, `us-3.platform.sh` . Para el resto de las indicaciones, puedes aceptar los valores predeterminados: un servidor en el plan de desarrollo más bajo ③ , tres entornos para el proyecto ④ , y 5 GB de almacenamiento para el proyecto global ⑤ .

The first prompt asks for a name for the project ① , so we use the name `ll_project` . The next prompt asks which region we'd like the server to be in ② . Choose the server closest to you; for me, that's `us-3.platform.sh` . For the rest of the prompts, you can accept the defaults: a server on the lowest development plan ③ , three environments for the project ④ , and 5GB of storage for the overall project ⑤ .

Hay tres preguntas más a las que responder:

There are three more prompts to respond to:

Default branch (--default-branch)

The default Git branch name for the project (the production environment)

Default: main

① > main

Git repository detected: `/Users/eric/.../learning_log`

② Set the new project `ll_project` as the remote for this repository? [Y/n] Y

The estimated monthly cost of this project is: \$10 USD

③ Are you sure you want to continue? [Y/n] Y

The Platform.sh Bot is activating your project



The project is now ready!

Un repositorio Git puede tener varias ramas; Platform.sh nos pregunta si la rama por defecto del proyecto debe ser `main` ❶ . Después nos pregunta si queremos conectar el repositorio local del proyecto al repositorio remoto ❷ . Por último, se nos informa de que este proyecto costará unos 10 \$ al mes si lo mantenemos en funcionamiento más allá del periodo de prueba gratuito ❸ . Si aún no has introducido una tarjeta de crédito, no deberías preocuparte por este coste. Platform.sh simplemente suspenderá tu proyecto si superas los límites de la prueba gratuita sin añadir una tarjeta de crédito.

A Git repository can have multiple branches; Platform.sh is asking us if the default branch for the project should be `main` ❶ . It then asks if we want to connect the local project's repository to the remote repository ❷ . Finally, we're informed that this project will cost about \$10 per month if we keep it running beyond the free trial period ❸ . If you haven't entered a credit card yet, you shouldn't have to worry about this cost. Platform.sh will simply suspend your project if you exceed the free trial's limits without adding a credit card.

Enviar a Plataforma.sh (*Pushing to Platform.sh*)

El último paso antes de ver la versión en vivo del proyecto es empujar nuestro código al servidor remoto. Para ello, ejecuta el siguiente comando:

The last step before seeing the live version of the project is to push our code to the remote server. To do that, issue the following command:

```
(ll_env)learning_log$ platform push
```

```
❶ Are you sure you want to push to the main (production) branch? [Y/n] Y
--snip--
```

```
The authenticity of host 'git.us-3.platform.sh (...)' can't be established.
RSA key fingerprint is SHA256:Tvn...7PM
```

```
❷ Are you sure you want to continue connecting (yes/no/[fingerprint])? Y
Pushing HEAD to the existing environment main
```

--snip--

```
To git.us-3.platform.sh:3pp3mqcexhlvy.git
```

```
* [new branch] HEAD -> main
```

Cuando emitas el comando `platform push`, se te pedirá una confirmación más de que quieres empujar el proyecto ❶. También puede que veas un mensaje sobre la autenticidad de Platform.sh, si es la primera vez que te conectas al sitio ❷. Introduce `Y` para cada uno de estos mensajes, y verás pasar un montón de resultados. Esta salida probablemente te parecerá confusa al principio, pero si algo va mal, te resultará muy útil durante la resolución de problemas. Si hojeas la salida, podrás ver dónde Platform.sh instala los paquetes necesarios, recopila archivos estáticos, aplica migraciones y configura URLs para el proyecto.

When you issue the command `platform push`, you'll be asked for one more confirmation that you want to push the project ❶. You may also see a message about the authenticity of Platform.sh, if this is your first time connecting to the site ❷. Enter `Y` for each of these prompts, and you'll see a bunch of output scroll by. This output will probably look confusing at first, but if anything goes wrong, it's really useful to have during troubleshooting. If you skim through the output, you can see where Platform.sh installs necessary packages, collects static files, applies migrations, and sets up URLs for the project.

Nota (Note)

Puede que veas un error de algo que puedes diagnosticar fácilmente, como una errata en uno de los archivos de configuración. Si esto ocurre, corrige el error en tu editor de texto, guarda el archivo y vuelve a ejecutar el comando `git commit`. Entonces podrás volver a ejecutar `platform push`.

You may see an error from something that you can easily diagnose, such as a typo in one of the configuration files. If this happens, fix the error in your text editor, save the file, and reissue the `git commit` command. Then you can run `platform push` again.

Ver el proyecto en vivo (Viewing the Live Project)

Una vez completado el push, puedes abrir el proyecto:

Once the push is complete, you can open the project:

```
(ll_env)learning_log$ platform url
Enter a number to open a URL
[0] https://main-bvx6a6i-wmye2fx7wwqgu.us-3.platformsh.site/
--snip--
> 0
```

El comando `platform url` enumera las URL asociadas a un proyecto desplegado; se te dará a elegir entre varias URL que son todas válidas para tu proyecto. Elige una y tu proyecto se abrirá en una nueva pestaña del navegador. Tendrá el mismo aspecto que el proyecto que hemos estado ejecutando localmente, pero puedes compartir esta URL con cualquier persona del mundo, y podrá acceder a tu proyecto y utilizarlo.

The `platform url` command lists the URLs associated with a deployed project; you'll be given a choice of several URLs that are all valid for your project. Choose one, and your project should open in a new browser tab! This will look just like the project we've been running locally, but you can share this URL with anyone in the world, and they can access and use your project.

Nota (Note)

Cuando despliegues tu proyecto utilizando una cuenta de prueba, no te sorprendas si a veces una página tarda más de lo normal en cargarse. En la mayoría de las plataformas de alojamiento, los recursos gratuitos que están inactivos suelen suspenderse y sólo se reinician cuando llegan nuevas solicitudes. La mayoría de las plataformas responden mucho mejor en los planes de alojamiento de pago.

When you deploy your project using a trial account, don't be surprised if it sometimes takes longer than usual for a page to load. On most hosting platforms, free resources that are idle are often suspended and only restarted when new requests come in. Most platforms are much more responsive on paid hosting plans.

Perfeccionar el despliegue de Platform.sh (Refining the Platform.sh Deployment)

Ahora perfeccionaremos el despliegue creando un superusuario, igual que hicimos localmente. También haremos que el proyecto sea más seguro cambiando la configuración `DEBUG` a `False`, para que los mensajes de error no muestren a los usuarios ninguna información adicional que puedan utilizar para atacar al servidor.

Now we'll refine the deployment by creating a superuser, just as we did locally. We'll also make the project more secure by changing the setting `DEBUG` to `False`, so error messages won't show users any extra information that they could use to attack the server.

Crear un superusuario en Plataforma.sh (Creating a Superuser on Platform.sh)

Se ha creado la base de datos del proyecto en vivo, pero está completamente vacía. Todos los usuarios que hemos creado antes sólo existen en nuestra versión local del proyecto.

The database for the live project has been set up, but it's completely empty. All the users we created earlier only exist in our local version of the project.

Para crear un superusuario en la versión en vivo del proyecto, iniciaremos una sesión SSH (secure socket shell) en la que podremos ejecutar comandos de gestión en el servidor remoto:

To create a superuser on the live version of the project, we'll start an SSH (secure socket shell) session where we can run management commands on the remote server:

```
(ll_env)learning_log$ platform environment:ssh
```

| _ \ | _ - | | _ / | _ _ _ _ _ _ _ _ | | _
 | _ / _ ' | _ | _ / _ \ ' | ' \ _ C - < ' \
 | _ | | _ \ , | _ \ | _ | _ \ _ | | _ | _ O _ | | _

Welcome to Platform.sh.

```
❶ web@ll_project.0:~$ ls
accounts learning_logs ll_project logs manage.py requirements.txt
requirements_remote.txt static
❷ web@ll_project.0:~$ python manage.py createsuperuser
❸ Username (leave blank to use 'web'): ll_admin_live
Email address:
Password:
Password (again):
Superuser created successfully.
❹ web@ll_project.0:~$ exit
logout
Connection to ssh.us-3.platform.sh closed.
❺ (ll_env)learning_log$
```

Cuando ejecutes por primera vez el comando `platform environment:ssh`, puede que aparezca otro mensaje sobre la autenticidad de este host. Si ves este mensaje, introduce `Y` y deberías acceder a una sesión de terminal remota.

When you first run the `platform environment:ssh` command, you may get another prompt about the authenticity of this host. If you see this message, enter `Y` and you should be logged in to a remote terminal session.

Después de ejecutar el comando `ssh`, tu terminal actúa igual que un terminal en el servidor remoto. Observa que tu prompt ha cambiado para indicar que estás en una sesión web asociada al proyecto llamado `ll_project` ❶. Si ejecutas el comando `ls`, verás los archivos que se han enviado al servidor Platform.sh.

After running the `ssh` command, your terminal acts just like a terminal on the remote server. Note that your prompt has changed to indicate that you're in a `web` session associated with the project named `ll_project` ❶. If you issue the `ls` command, you'll see the files that have been pushed to the Platform.sh server.

Emite el mismo comando `createsuperuser` que utilizamos en el Capítulo 18 ❷. Esta vez, he introducido un nombre de usuario de administrador, `ll_admin_live`, que es distinto del que utilicé localmente ❸. Cuando hayas terminado de trabajar en la sesión de terminal remota, introduce el comando `exit` ❹. Tu indicador te dirá que estás trabajando de

nuevo en tu sistema local ❸ .

Issue the same `createsuperuser` command we used in Chapter 18 ❷ . This time, I entered an admin username, `ll_admin_live` , that's distinct from the one I used locally ❸ . When you're finished working in the remote terminal session, enter the `exit` command ❹ . Your prompt will indicate that you're working in your local system again ❺ .

Ahora puedes añadir `/admin/` al final de la URL de la aplicación en vivo e iniciar sesión en el sitio de administración. Si otras personas ya han empezado a utilizar tu proyecto, ¡ten en cuenta que tendrás acceso a todos sus datos! Tómate en serio esta responsabilidad y los usuarios seguirán confiándote sus datos.

Now you can add `/admin/` to the end of the URL for the live app and log in to the admin site. If others have already started using your project, be aware that you'll have access to all their data! Take this responsibility seriously, and users will continue to trust you with their data.

Nota (Note)

Los usuarios de Windows utilizarán los mismos comandos que se muestran aquí (como `ls` en lugar de `dir`), porque estás ejecutando un terminal Linux a través de una conexión remota.

Windows users will use the same commands shown here (such as `ls` instead of `dir`), because you're running a Linux terminal through a remote connection.

Asegurar el proyecto en vivo (Securing the Live Project)

Hay un problema de seguridad evidente en la forma en que se despliega actualmente nuestro proyecto: la configuración `DEBUG = True` en `settings.py` , que proporciona mensajes de depuración cuando se producen errores. Las páginas de error de Django te proporcionan información de depuración vital cuando estás desarrollando un proyecto; sin embargo, dan demasiada información a los atacantes si las dejas activadas en un servidor

activo.

There's one glaring security issue in the way our project is currently deployed: the setting `DEBUG = True` in *settings.py*, which provides debug messages when errors occur. Django's error pages give you vital debugging information when you're developing a project; however, they give away too much information to attackers if you leave them enabled on a live server.

Para ver lo malo que es esto, ve a la página de inicio de tu proyecto desplegado. Accede a la cuenta de un usuario y añade `/topics/999/` al final de la URL de la página de inicio. Suponiendo que no hayas hecho miles de temas, deberías ver una página con el mensaje *DoesNotExist at /topics/999/*. Si te desplazas hacia abajo, deberías ver un montón de información sobre el proyecto y el servidor. No querrás que tus usuarios vean esto, y desde luego no querrás que esta información esté disponible para cualquier persona interesada en atacar el sitio.

To see how bad this is, go to the home page of your deployed project. Log in to a user's account and add `/topics/999/` to the end of the home page URL. Assuming you haven't made thousands of topics, you should see a page with the message *DoesNotExist at /topics/999/*. If you scroll down, you should see a whole bunch of information about the project and the server. You won't want your users to see this, and you certainly wouldn't want this information available to anyone interested in attacking the site.

Podemos evitar que esta información se muestre en el sitio en vivo configurando `DEBUG = False` en la parte de *settings.py* que sólo se aplica a la versión desplegada del proyecto. De este modo, seguirás viendo la información de depuración localmente, donde esa información es útil, pero no se mostrará en el sitio en vivo.

We can prevent this information from being shown on the live site by setting `DEBUG = False` in the part of *settings.py* that only applies to the deployed version of the project. This way you'll continue to see debugging information locally, where that information is useful, but it won't show up on the live site.

Abre *settings.py* en tu editor de texto, y añade una línea de código a la parte que modifica la configuración de `Platform.sh`:

Open *settings.py* in your text editor, and add one line of code to the part that modifies settings for Platform.sh:

settings.py

settings.py

```
--snip--
if config.is_valid_platform():
    ALLOWED_HOSTS.append('.platformsh.site')
    DEBUG = False
--snip--
```

Todo el trabajo para establecer la configuración de la versión desplegada del proyecto ha dado sus frutos. Cuando queramos ajustar la versión en vivo del proyecto, sólo tenemos que cambiar la parte pertinente de la configuración que establecimos antes.

All the work to set up configuration for the deployed version of the project has paid off. When we want to adjust the live version of the project, we just change the relevant part of the configuration we set up earlier.

Confirmar y enviar cambios (Committing and Pushing Changes)

Ahora tenemos que confirmar los cambios realizados en *settings.py* y enviar los cambios a Platform.sh. Aquí tienes una sesión de terminal que muestra la primera parte de este proceso:

Now we need to commit the changes made to *settings.py* and push the changes to Platform.sh. Here's a terminal session showing the first part of this process:

```
❶ (ll_env)learning_log$ git commit -am "Set DEBUG False on live site."
[main d2ad0f7] Set DEBUG False on live site.
1 file changed, 1 insertion(+)
```

```
❷ (ll_env)learning_log$ git status
On branch main
nothing to commit, working tree clean
(ll_env)learning_log$
```

Emitimos el comando `git commit` con un mensaje de confirmación breve pero descriptivo ❶. Recuerda que la bandera `-am` se asegura de que Git confirma todos los archivos que han cambiado y registra el mensaje de registro. Git reconoce que un archivo ha cambiado y consigna este cambio en el repositorio.

We issue the `git commit` command with a short but descriptive commit message ❶. Remember the `-am` flag makes sure Git commits all the files that have changed and records the log message. Git recognizes that one file has changed and commits this change to the repository.

Ejecutar `git status` muestra que estamos trabajando en la rama `main` del repositorio y que ahora no hay nuevos cambios que confirmar ❷. Es importante comprobar el estado antes de enviar a un servidor remoto. Si no ves un estado limpio, entonces algunos cambios no se han confirmado y no se enviarán al servidor. Puedes intentar emitir de nuevo el comando `commit`; si no estás seguro de cómo resolver el problema, lee el Apéndice D para comprender mejor cómo trabajar con Git.

Running `git status` shows that we're working on the `main` branch of the repository and that there are now no new changes to commit ❷. It's important to check the status before pushing to a remote server. If you don't see a clean status, then some changes haven't been committed and those changes won't be pushed to the server. You can try issuing the `commit` command again; if you're not sure how to resolve the issue, read through Appendix D to better understand how to work with Git.

Ahora vamos a empujar el repositorio actualizado a Plataforma.sh:

Now let's push the updated repository to Platform.sh:

```
(ll_env)learning_log$ platform push
Are you sure you want to push to the main (production) branch? [Y/n] Y
Pushing HEAD to the existing environment main
```

--snip--

```
To git.us-3.platform.sh:wmye2fx7wwqgu.git  
fce0206..d2ad0f7 HEAD -> main  
(ll_env)learning_log$
```

Platform.sh reconoce que el repositorio se ha actualizado, y reconstruye el proyecto para asegurarse de que se han tenido en cuenta todos los cambios. No reconstruye la base de datos, por lo que no hemos perdido ningún dato.

Platform.sh recognizes that the repository has been updated, and it rebuilds the project to make sure all the changes have been taken into account. It doesn't rebuild the database, so we haven't lost any data.

Para asegurarte de que este cambio ha surtido efecto, visita de nuevo la URL `/topics/999/`. Sólo deberías ver el mensaje *Server Error (500)*, sin ningún tipo de información sensible sobre el proyecto.

To make sure this change took effect, visit the `/topics/999/` URL again. You should see just the message *Server Error (500)*, with no sensitive information about the project at all.

Crear páginas de error personalizadas (Creating Custom Error Pages)

En el Capítulo 19, configuramos el Registro de Aprendizaje para que devolviera un error 404 si el usuario solicitaba un tema o una entrada que no le pertenecía. Ahora también has visto un error de servidor 500. Un error 404 suele significar que tu código Django es correcto, pero el objeto solicitado no existe. Un error 500 suele significar que hay un error en el código que has escrito, como un error en una función en `views.py`. Actualmente, Django devuelve la misma página de error genérica en ambas situaciones, pero podemos escribir nuestras propias plantillas de páginas de error 404 y 500 que coincidan con la apariencia general de Learning Log. Estas plantillas pertenecen al directorio raíz de plantillas.

In Chapter 19, we configured Learning Log to return a 404 error if the user requests a topic or entry that doesn't belong to them. Now you've seen a 500 server error as well. A 404 error usually means your Django code is correct, but the object being requested doesn't exist. A 500 error usually means there's an error in the code you've written, such as an error in a function in `views.py`. Django currently returns the same generic error page in

both situations, but we can write our own 404 and 500 error page templates that match Learning Log's overall appearance. These templates belong in the root template directory.

Crear plantillas personalizadas (Making Custom Templates)

En la carpeta *learning_log* , crea una nueva carpeta llamada *templates* . Luego crea un nuevo archivo llamado *404.xhtml* ; la ruta a este archivo debe ser *learning_log/templates/404.xhtml* . Aquí tienes el código de este archivo:

In the *learning_log* folder, make a new folder called *templates* . Then make a new file called *404.xhtml* ; the path to this file should be *learning_log/templates/404.xhtml* . Here's the code for this file:

404.xhtml

404.xhtml

```
{% extends "learning_logs/base.xhtml" %}
```

```
{% block page_header %}
```

```
<h2>The item you requested is not available. (404)</h2>
```

```
{% endblock page_header %}
```

Esta sencilla plantilla proporciona la información genérica de la página de error 404, pero su estilo se adapta al del resto del sitio.

This simple template provides the generic 404 error page information but is styled to match the rest of the site.

Crea otro archivo llamado *500.xhtml* utilizando el siguiente código:

Make another file called *500.xhtml* using the following code:

500.xhtml

500.xhtml

```
{% extends "learning_logs/base.xhtml" %}

{% block page_header %}
    <h2>There has been an internal error. (500)</h2>
{% endblock page_header %}
```

Estos nuevos archivos requieren un ligero cambio en *settings.py* .

These new files require a slight change to *settings.py* .

ajustes.py

settings.py

```
--snip--
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [BASE_DIR / 'templates'],
        'APP_DIRS': True,
        --snip--
    },
]
--snip--
```

Este cambio indica a Django que busque en el directorio raíz de plantillas las plantillas de páginas de error y cualquier otra plantilla que no esté asociada a una aplicación concreta.

This change tells Django to look in the root template directory for the error page templates and any other templates that aren't associated with a particular app.

Enviar los cambios a Platform.sh (Pushing the Changes to Platform.sh)

Ahora tenemos que confirmar los cambios que acabamos de hacer y enviarlos a Platform.sh:

Now we need to commit the changes we just made and push them to Platform.sh:

```
❶ (ll_env)learning_log$ git add .
❷ (ll_env)learning_log$ git commit -am "Added custom 404 and 500 error pages."
3 files changed, 11 insertions(+), 1 deletion(-)
create mode 100644 templates/404.xhtml
create mode 100644 templates/500.xhtml
❸ (ll_env)learning_log$ platform push
--snip--
To git.us-3.platform.sh:wmye2fx7wwqgu.git
d2ad0f7..9f042ef HEAD -> main
(ll_env)learning_log$
```

Emitimos el comando `git add .` ❶ porque hemos creado algunos archivos nuevos en el proyecto. Luego confirmamos los cambios ❷ y enviamos el proyecto actualizado a Plataforma.sh ❸ .

We issue the `git add .` command ❶ because we created some new files in the project. Then we commit the changes ❷ and push the updated project to Platform.sh ❸ .

Ahora, cuando aparezca una página de error, tendrá el mismo estilo que el resto del sitio, lo que facilitará la experiencia del usuario cuando se produzcan errores.

Now when an error page appears, it should have the same styling as the rest of the site, making for a smoother user experience when errors arise.

Desarrollo en curso (Ongoing Development)

Es posible que quieras seguir desarrollando Learning Log después de su lanzamiento inicial en un servidor activo, o que quieras desarrollar tus propios proyectos para desplegarlos. Al

hacerlo, existe un proceso bastante coherente para actualizar tus proyectos.

You might want to further develop Learning Log after your initial push to a live server, or you might want to develop your own projects to deploy. When doing so, there's a fairly consistent process for updating your projects.

En primer lugar, realizarás los cambios necesarios en tu proyecto local. Si tus cambios dan lugar a algún archivo nuevo, añade esos archivos al repositorio Git utilizando el comando `git add .` (asegurándote de incluir el punto al final del comando). Cualquier cambio que requiera una migración de la base de datos necesitará este comando, porque cada migración genera un nuevo archivo de migración.

First, you'll make the necessary changes to your local project. If your changes result in any new files, add those files to the Git repository using the command `git add .` (making sure to include the dot at the end of the command). Any change that requires a database migration will need this command, because each migration generates a new migration file.

En segundo lugar, confirma los cambios en tu repositorio utilizando `git commit -am "commit message"`. A continuación, empuja tus cambios a Plataforma.sh, utilizando el comando `platform push`. Visita tu proyecto en vivo y asegúrate de que los cambios que esperas ver han surtido efecto.

Second, commit the changes to your repository using `git commit -am "commit message"`. Then push your changes to Platform.sh, using the command `platform push`. Visit your live project and make sure the changes you expect to see have taken effect.

Es fácil cometer errores durante este proceso, así que no te sorprendas cuando algo vaya mal. Si el código no funciona, revisa lo que has hecho e intenta detectar el error. Si no encuentras el error o no sabes cómo deshacerlo, consulta las sugerencias para obtener ayuda en el Apéndice C. No seas tímido a la hora de pedir ayuda: todo el mundo aprendió a construir proyectos haciéndose las mismas preguntas que probablemente te harás tú, así que alguien estará encantado de ayudarte. Resolver cada problema que surja te ayudará a desarrollar constantemente tus habilidades hasta que estés construyendo proyectos significativos y fiables y respondiendo también a las preguntas de otras personas.

It's easy to make mistakes during this process, so don't be surprised when something goes wrong. If the code doesn't work, review what you've done and try to spot the mistake. If you can't find the mistake or you can't figure out how to undo it, refer to the suggestions for getting help in [Appendix C](#) . Don't be shy about asking for help: everyone else learned to build projects by asking the same questions you're likely to ask, so someone will be happy to help you. Solving each problem that arises helps you steadily develop your skills until you're building meaningful, reliable projects and answering other people's questions as well.

Eliminar un proyecto en Plataforma.sh *(Deleting a Project on Platform.sh)*

Es una buena práctica realizar el proceso de despliegue varias veces con el mismo proyecto o con una serie de proyectos pequeños, para cogerle el truco al despliegue. Pero necesitarás saber cómo eliminar un proyecto que se ha desplegado. Platform.sh también limita el número de proyectos que puedes alojar gratuitamente, y no querrás saturar tu cuenta con proyectos de práctica.

It's great practice to run through the deployment process a number of times with the same project or with a series of small projects, to get the hang of deployment. But you'll need to know how to delete a project that's been deployed. Platform.sh also limits the number of projects you can host for free, and you don't want to clutter your account with practice projects.

Puedes eliminar un proyecto utilizando la CLI:

You can delete a project using the CLI:

```
(ll_env)learning_log$ platform project:delete
```

Se te pedirá que confirmes que deseas realizar esta acción destructiva. Responde a las indicaciones y tu proyecto será eliminado.

You'll be asked to confirm that you want to take this destructive action. Respond to the prompts, and your project will be deleted.

El comando `platform create` también dio al repositorio Git local una referencia al repositorio remoto en los servidores de Platform.sh. También puedes eliminar este remoto desde la línea de comandos:

The command `platform create` also gave the local Git repository a reference to the remote repository on Platform.sh's servers. You can remove this remote from the command line as well:

```
(ll_env)learning_log$ git remote
platform
(ll_env)learning_log$ git remote remove platform
```

El comando `git remote` lista los nombres de todas las URL remotas asociadas al repositorio actual. El comando `git remote remove remote_name` elimina estas URL remotas del repositorio local.

The command `git remote` lists the names of all remote URLs associated with the current repository. The command `git remote remove remote_name` deletes these remote URLs from the local repository.

También puedes eliminar los recursos de un proyecto accediendo a la página web Platform.sh y visitando tu panel de control en <https://console.platform.sh>. Esta página enumera todos tus proyectos activos. Haz clic en los tres puntos de la casilla de un proyecto, y haz clic en **Editar Plan**. Esta es una página de precios para el proyecto; haz clic en el botón **Eliminar proyecto** en la parte inferior de la página, y se te mostrará una página de confirmación en la que podrás llevar a cabo la eliminación. Aunque hayas eliminado tu proyecto utilizando la CLI, es una buena idea que te familiarices con el panel de control de cualquier proveedor de alojamiento en el que te despliegues.

You can also delete a project's resources by logging in to the Platform.sh website and visiting your dashboard at <https://console.platform.sh>. This page lists all your active projects. Click the three dots in a project's box, and click **Edit Plan**. This is a pricing page for the project; click the **Delete Project** button at the bottom of the page, and you'll be shown a confirmation page where you can follow through with the deletion. Even if you deleted your project using the CLI, it's a good idea to familiarize yourself with the dashboard of any hosting provider you deploy to.

Nota (Note)

Eliminar un proyecto en Platform.sh no hace nada a tu versión local del proyecto. Si nadie ha utilizado tu proyecto desplegado y sólo estás practicando el proceso de despliegue, es perfectamente razonable borrar tu proyecto en Platform.sh y volver a desplegarlo. Sólo ten en cuenta que si las cosas dejan de funcionar, puede que te hayas topado con las limitaciones de nivel libre del host.

Deleting a project on Platform.sh does nothing to your local version of the project. If no one has used your deployed project and you're just practicing the deployment process, it's perfectly reasonable to delete your project on Platform.sh and redeploy it. Just be aware that if things stop working, you may have run into the host's free-tier limitations.

Pruébalo tú mismo (Try It Yourself)

20-3. Blog en vivo: Despliega el proyecto Blog en el que has estado trabajando en Plataforma.sh. Asegúrate de configurar `DEBUG` en `False`, para que los usuarios no vean las páginas de error completas de Django cuando algo vaya mal.

20-3. Live Blog: Deploy the Blog project you've been working on to Platform.sh. Make sure you set `DEBUG` to `False`, so users don't see the full Django error pages when something goes wrong.

20-4. Registro de aprendizaje ampliado: Añade una característica al Registro de Aprendizaje, y empuja el cambio a tu despliegue en vivo. Prueba con un cambio sencillo, como escribir más sobre el proyecto en la página de inicio. Después, prueba a añadir una función más avanzada, como dar a los usuarios la opción de hacer público un tema. Esto requeriría un atributo llamado `public` como parte del modelo `Topic` (debería estar configurado como `False` por defecto) y un elemento de formulario en la página `new_topic` que permita al usuario cambiar un tema de privado a público. Entonces tendrías que migrar el proyecto y revisar `views.py` para que cualquier tema que sea público también sea visible para los usuarios no autenticados.

20-4. Extended Learning Log: Add one feature to Learning Log, and push the change to your live deployment. Try a simple change, such as writing more about the project on the home page. Then try adding a more advanced feature, such as giving users the option of making a topic public. This would require an attribute called `public` as part of the `Topic` model (this should be set to `False` by default) and a form element on the `new_topic` page that allows the user to change a topic from private to public. You'd then need to migrate the project and revise `views.py` so any topic that's public is visible to unauthenticated users as well.

Resumen (Summary)

En este capítulo has aprendido a dar a tus proyectos un aspecto sencillo pero profesional utilizando la biblioteca Bootstrap y la aplicación `django-bootstrap5`. Con Bootstrap, los estilos que elijas funcionarán de forma consistente en casi cualquier dispositivo que la gente utilice para acceder a tu proyecto.

In this chapter, you learned to give your projects a simple but professional appearance using the Bootstrap library and the `django-bootstrap5` app. With Bootstrap, the styles you choose will work consistently on almost any device people use to access your project.

Has aprendido sobre las plantillas de Bootstrap y has utilizado la plantilla *Navbar static* para crear un aspecto sencillo para Learning Log. Utilizaste un jumbotróon para destacar el mensaje de una página de inicio, y aprendiste a aplicar un estilo coherente a todas las páginas de un sitio.

You learned about Bootstrap's templates and used the *Navbar static* template to create a simple look and feel for Learning Log. You used a jumbotron to make a home page's message stand out, and learned to style all the pages in a site consistently.

En la parte final del proyecto, aprendiste a desplegar un proyecto en un servidor remoto para que cualquiera pueda acceder a él. Creaste una cuenta Platform.sh e instalaste algunas herramientas que ayudan a gestionar el proceso de despliegue. Utilizaste Git para enviar el proyecto de trabajo a un repositorio, y luego enviaste el repositorio a un servidor remoto en Platform.sh. Por último, aprendiste a proteger tu aplicación configurando `DEBUG = False` en el servidor activo. También creaste páginas de error personalizadas, para que los inevitables errores que surjan se vean bien gestionados.

In the final part of the project, you learned how to deploy a project to a remote server so anyone can access it. You made a Platform.sh account and installed some tools that help manage the deployment process. You used Git to commit the working project to a repository, and then pushed the repository to a remote server on Platform.sh. Finally, you learned to begin securing your app by setting `DEBUG = False` on the live server. You also made custom error pages, so the inevitable errors that come up will look well-handled.

Ahora que has terminado el Registro de Aprendizaje, puedes empezar a construir tus propios proyectos. Empieza de forma sencilla y asegúrate de que el proyecto funciona antes de añadirle complejidad. Disfruta de tu aprendizaje continuo, ¡y buena suerte con tus proyectos!

Now that you've finished Learning Log, you can start building your own projects. Start simple, and make sure the project works before adding complexity. Enjoy your continued learning, and good luck with your projects!

Apéndice A: Instalación y resolución de problemas (Appendix A: Installation and Troubleshooting)

A

Instalación y resolución de problemas (*A Installation and Troubleshooting*)



Hay muchas versiones de Python disponibles y numerosas formas de configurarlo en cada sistema operativo. Si el método del Capítulo 1 no ha funcionado, o si quieres instalar una versión de Python distinta de la que tienes instalada, las instrucciones de este apéndice pueden ayudarte.

There are many versions of Python available and numerous ways to set it up on each operating system. If the approach in Chapter 1 didn't work, or if you want to install a different version of Python than the one currently installed, the instructions in this appendix can help.

Python en Windows (*Python on Windows*)

Las instrucciones del Capítulo 1 te muestran cómo instalar Python utilizando el instalador oficial en <https://python.org>. Si no has conseguido que Python se ejecute después de utilizar el instalador, las instrucciones de resolución de problemas de esta sección deberían ayudarte a poner Python en marcha.

The instructions in Chapter 1 show you how to install Python using the official installer at <https://python.org>. If you couldn't get Python to run after using the installer, the troubleshooting instructions in this section should help you get Python up and running.

Utilizar `py` en lugar de `python` (*Using `py` Instead of `python`*)

Si ejecutas un instalador reciente de Python y luego emites el comando `python` en un terminal, deberías ver el prompt de Python para una sesión de terminal (`>>>`). Cuando Windows no reconozca el comando `python`, abrirá la Microsoft Store porque cree que Python no está instalado, o recibirás un mensaje como "No se encontró Python". Si se abre el Microsoft Store, ciérralo; es mejor utilizar el instalador oficial de Python de <https://python.org> que el que mantiene Microsoft.

If you run a recent Python installer and then issue the command `python` in a terminal, you should see the Python prompt for a terminal session (`>>>`). When Windows doesn't recognize the `python` command, it will either open the Microsoft Store because it thinks Python isn't installed, or you'll get a message such as "Python was not found." If the Microsoft Store opens, close it; it's better to use the official Python installer from <https://python.org> than the one that Microsoft maintains.

La solución más sencilla, sin hacer ningún cambio en tu sistema, es probar el comando `py`. Se trata de una utilidad de Windows que encuentra la última versión de Python instalada en tu sistema y ejecuta ese intérprete. Si este comando funciona y quieres utilizarlo, simplemente utiliza `py` en cualquier lugar donde veas el comando `python` o `python3` en este libro.

The simplest solution, without making any changes to your system, is to try the `py` command. This is a Windows utility that finds the latest version of Python installed on your system and runs that interpreter. If this command works and you want to use it, simply use `py` anywhere you see the `python` or `python3` command in this book.

Volver a ejecutar el instalador (*Rerunning the Installer*)

La razón más común por la que `python` no funciona es que la gente olvida seleccionar la opción Añadir Python al PATH al ejecutar el instalador; es un error fácil de cometer. La variable `PATH` es una configuración del sistema que indica a Python dónde buscar los programas más utilizados. En este caso, Windows no sabe cómo encontrar el intérprete de Python.

The most common reason `python` doesn't work is that people forget to select the Add Python to PATH option when running the installer; this is an easy mistake to make. The `PATH` variable is a system setting that tells Python where to look for commonly used programs. In this case, Windows doesn't know how to find the Python interpreter.

La solución más sencilla en esta situación es volver a ejecutar el instalador. Si hay un instalador más reciente disponible en <https://python.org>, descarga el nuevo instalador y ejecútalo, asegurándote de marcar la casilla **Añadir Python a PATH**.

The simplest fix in this situation is to run the installer again. If there's a newer installer available from <https://python.org>, download the new installer and run it, making sure to check the **Add Python to PATH** box.

Si ya tienes el último instalador, ejecútalo de nuevo y selecciona la opción **Modificar**. Verás una lista de características opcionales; mantén seleccionadas las opciones por defecto en esta pantalla. A continuación, haz clic en **Siguiente** y marca la casilla **Añadir Python a las variables de entorno**. Por último, haz clic en **Instalar**. El instalador reconocerá que Python ya está instalado, y añadirá la ubicación del intérprete de Python a la variable `PATH`. Asegúrate de cerrar los terminales que tengas abiertos, porque seguirán utilizando la antigua variable `PATH`. Abre una nueva ventana de terminal y emite de nuevo el comando `python`; deberías ver un prompt de Python (`>>>`).

If you already have the latest installer, run it again and select the **Modify** option. You'll see a list of optional features; keep the default options selected on this screen. Then click **Next** and check the **Add Python to Environment Variables** box. Finally, click **Install**. The installer will recognize that Python is already installed, and it will add the location of the Python interpreter to the `PATH` variable. Make sure you close any open terminals, because

they'll still be using the old `PATH` variable. Open a new terminal window and issue the command `python` again; you should see a Python prompt (`>>>`).

Python en macOS (*Python on macOS*)

Las instrucciones de instalación del Capítulo 1 utilizan el instalador oficial de Python en <https://python.org>. El instalador oficial funciona bien desde hace años, pero hay algunas cosas que pueden despistarte. Esta sección te ayudará si algo no funciona de forma sencilla.

The installation instructions in Chapter 1 use the official Python installer at <https://python.org>. The official installer has been working well for years now, but there are a few things that can get you off track. This section will help if anything isn't working in a straightforward manner.

Instalar accidentalmente la versión de Python de Apple (*Accidentally Installing Apple's Version of Python*)

Si ejecutas el comando `python3` y Python aún no está instalado en tu sistema, lo más probable es que aparezca un mensaje indicando que es necesario instalar *command line developer tools*. Lo mejor en este momento es cerrar la ventana emergente que muestra este mensaje, descargar el instalador de Python desde <https://python.org>, y ejecutar el instalador.

If you run the `python3` command and Python is not yet installed on your system, you'll most likely see a message that the *command line developer tools* need to be installed. The best approach at this point is to close the pop-up showing this message, download the Python installer from <https://python.org>, and run the installer.

Si decides instalar las herramientas de desarrollador de línea de comandos en este punto, macOS instalará la versión de Apple de Python junto con las herramientas de desarrollador. El único problema de esto es que la versión de Python de Apple suele ir algo por detrás de la última versión oficial de Python. Sin embargo, puedes descargar y ejecutar el instalador oficial desde <https://python.org>, y `python3` apuntará entonces a la versión más reciente. No te preocupes por tener instaladas las herramientas para desarrolladores; hay algunas herramientas útiles, como el sistema de control de versiones Git, del que se habla en el Apéndice D.

If you choose to install the command line developer tools at this point, macOS will install Apple's version of Python along with the developer tools. The only issue with this is that Apple's version of Python is usually somewhat behind the latest official version of Python. However, you can still download and run the official installer from <https://python.org>, and `python3` will then point to the newer version. Don't worry about having the developer tools installed; there are some useful tools in there, including the Git version control system discussed in Appendix D.

Python 2 en versiones antiguas de macOS (*Python 2 on Older Versions of macOS*)

En versiones antiguas de macOS, anteriores a Monterey (macOS 12), se instalaba por defecto una versión obsoleta de Python 2. En estos sistemas, el comando `python` apunta al intérprete anticuado del sistema. Si utilizas una versión de macOS con Python 2 instalado, asegúrate de utilizar el comando `python3`, y siempre utilizarás la versión de Python que hayas instalado.

On older versions of macOS, before Monterey (macOS 12), an outdated version of Python 2 was installed by default. On these systems, the command `python` points to the outdated system interpreter. If you're using a version of macOS with Python 2 installed, make sure you use the `python3` command, and you'll always be using the version of Python you installed.

Python en Linux (*Python on Linux*)

Python está incluido por defecto en casi todos los sistemas Linux. Sin embargo, si la versión por defecto de tu sistema es anterior a Python 3.9, debes instalar la última versión. También puedes instalar la última versión si quieres las características más recientes, como los mensajes de error mejorados de Python. Las siguientes instrucciones deberían funcionar para la mayoría de los sistemas basados en apt.

Python is included by default on almost every Linux system. However, if the default version on your system is earlier than Python 3.9, you should install the latest version. You can also install the latest version if you want the most recent features, like Python's improved error messages. The following instructions should work for most apt-based systems.

Utilizar la instalación por defecto de Python (*Using the Default Python Installation*)

Si quieres utilizar la versión de Python a la que apunta `python3`, asegúrate de tener instalados estos tres paquetes adicionales:

If you want to use the version of Python that `python3` points to, make sure you have these three additional packages installed:

```
$ sudo apt install python3-dev python3-pip python3-venv
```

Estos paquetes incluyen herramientas útiles para desarrolladores y herramientas que te permiten instalar paquetes de terceros, como los utilizados en la sección de proyectos de este libro.

These packages include tools that are useful for developers and tools that let you install third-party packages, like the ones used in the projects section of this book.

Instalar la última versión de Python *(Installing the Latest Version of Python)*

Utilizaremos un paquete llamado `deadsnakes`, que facilita la instalación de varias versiones de Python. Introduce los siguientes comandos:

We'll use a package called `deadsnakes`, which makes it easy to install multiple versions of Python. Enter the following commands:

```
$ sudo add-apt-repository ppa:deadsnakes/ppa
$ sudo apt update
$ sudo apt install python3.11
```

Estos comandos instalarán Python 3.11 en tu sistema.

These commands will install Python 3.11 onto your system.

Introduce el siguiente comando para iniciar una sesión de terminal que ejecute Python 3.11:

Enter the following command to start a terminal session that runs Python 3.11:

```
$ python3.11  
>>>
```

En cualquier lugar donde veas el comando `python` en este libro, utiliza en su lugar `python3.11`. También querrás utilizar este comando cuando ejecutes programas desde el terminal.

Anywhere you see the command `python` in this book, use `python3.11` instead. You'll also want to use this command when you run programs from the terminal.

Necesitarás instalar dos paquetes más para sacar el máximo partido a tu instalación de Python:

You'll need to install two more packages to make the most of your Python installation:

```
$ sudo apt install python3.11-dev python3.11-venv
```

Estos paquetes incluyen módulos que necesitarás al instalar y ejecutar paquetes de terceros, como los utilizados en los proyectos de la segunda mitad del libro.

These packages include modules you'll need when installing and running third-party packages, like the ones used in the projects in the second half of the book.

Nota (Note)

El paquete `deadsnakes` se ha mantenido activamente durante mucho tiempo. Cuando salgan nuevas versiones de Python, puedes utilizar estos mismos comandos, sustituyendo `python3.11` por la última versión disponible en ese momento.

The `deadsnakes` package has been actively maintained for a long time. When newer versions of Python come out, you can use these same commands, replacing `python3.11` with the latest version currently available.

Comprobar qué versión de Python estás utilizando (*Checking Which Version of Python You're Using*)

Si tienes algún problema al ejecutar Python o al instalar paquetes adicionales, puede ser útil saber exactamente qué versión de Python estás utilizando. Puedes tener varias versiones de Python instaladas y no tener claro qué versión se está utilizando actualmente.

If you're having any issues running Python or installing additional packages, it can be helpful to know exactly which version of Python you're using. You may have multiple versions of Python installed and not be clear about which version is currently being used.

Ejecuta el siguiente comando en un terminal:

Issue the following command in a terminal:

```
$ python --version
Python 3.11.0
```

Esto te dice exactamente a qué versión está apuntando actualmente el comando `python`. El comando más corto `python -V` dará el mismo resultado.

This tells you exactly which version the command `python` is currently pointing to. The shorter command `python -V` will give the same output.

Palabras clave y funciones incorporadas de Python (*Python Keywords and Built-in Functions*)

Python viene con su propio conjunto de palabras clave y funciones incorporadas. Es importante tenerlas en cuenta cuando nombres cosas en Python: tus nombres no pueden

coincidir con estas palabras clave y no deben coincidir con los nombres de las funciones, o sobrescribirás las funciones.

Python comes with its own set of keywords and built-in functions. It's important to be aware of these when you're naming things in Python: your names cannot be the same as these keywords and shouldn't be the same as the function names, or you'll overwrite the functions.

En esta sección, enumeraremos las palabras clave de Python y los nombres de las funciones incorporadas, para que sepas qué nombres debes evitar.

In this section, we'll list Python's keywords and built-in function names, so you'll know which names to avoid.

Palabras clave de Python (*Python Keywords*)

Cada una de las siguientes palabras clave tiene un significado específico, y verás un error si intentas utilizar cualquiera de ellas como nombre de variable.

Each of the following keywords has a specific meaning, and you'll see an error if you try to use any of them as a variable name.

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

Funciones incorporadas de Python (*Python Built-in Functions*)

No obtendrás un error si utilizas una de las siguientes funciones incorporadas fácilmente disponibles como nombre de variable, pero anularás el comportamiento de esa función:

You won't get an error if you use one of the following readily available built-in functions as a variable name, but you'll override the behavior of that function:

abs()	hash()	slice()
aiter()	help()	sorted()
all()	hex()	staticmethod()
any()	id()	str()
anext()	input()	sum()
ascii()	int()	super()
bin()	isinstance()	tuple()
bool()	issubclass()	type()
breakpoint()	iter()	vars()
bytearray()	len()	zip()
bytes()	list()	__import__()
callable()	locals()	
chr()	map()	
classmethod()	max()	
compile()	memoryview()	
complex()	min()	
delattr()	next()	
dict()	object()	
dir()	oct()	
divmod()	open()	
enumerate()	ord()	
eval()	pow()	
exec()	print()	
filter()	property()	
float()	range()	
format()	repr()	
frozenset()	reversed()	
getattr()	round()	
globals()	set()	
hasattr()	setattr()	

Apéndice B: Editores de texto e IDEs (Appendix B: Text Editors and IDEs)

B

Editores de texto e IDEs (*B Text Editors and IDEs*)



Los programadores pasan mucho tiempo escribiendo, leyendo y editando código, y es esencial utilizar un editor de texto o un IDE (entorno de desarrollo integrado) para que este trabajo sea lo más eficiente posible. Un buen editor realizará tareas sencillas, como resaltar la estructura de tu código para que puedas detectar errores comunes mientras trabajas. Pero no hará tanto que te distraiga de tus pensamientos. Los editores también tienen funciones útiles como la sangría automática, marcadores para mostrar la longitud de línea adecuada y atajos de teclado para operaciones comunes.

Programmers spend a lot of time writing, reading, and editing code, and using a text editor or an IDE (integrated development environment) to make this work as efficient as possible is essential. A good editor will do simple tasks, like highlighting your code's structure so you can catch common bugs as you're working. But it won't do so much that it distracts you

from your thinking. Editors also have useful features like automatic indenting, markers to show appropriate line length, and keyboard shortcuts for common operations.

Un *IDE* es un editor de texto con otras herramientas incluidas, como depuradores interactivos e introspección de código. Un IDE examina tu código a medida que lo introduces e intenta aprender sobre el proyecto que estás construyendo. Por ejemplo, cuando empiezas a escribir el nombre de una función, un IDE puede mostrarte todos los argumentos que acepta esa función. Este comportamiento puede ser muy útil cuando todo funciona y entiendes lo que estás viendo. Pero también puede resultar abrumador para un principiante y difícil de solucionar cuando no estás seguro de por qué tu código no funciona en el IDE.

An *IDE* is a text editor with a number of other tools included, like interactive debuggers and code introspection. An IDE examines your code as you enter it and tries to learn about the project you're building. For example, when you start typing the name of a function, an IDE might show you all the arguments that function accepts. This behavior can be very helpful when everything works and you understand what you're seeing. But it can also be overwhelming as a beginner and difficult to troubleshoot when you aren't sure why your code isn't working in the IDE.

Hoy en día, las líneas entre los editores de texto y los IDE se han difuminado. Los editores más populares tienen algunas funciones que antes eran exclusivas de los IDE. Asimismo, la mayoría de los IDE pueden configurarse para funcionar en un modo más ligero que te distraiga menos mientras trabajas, pero que te permita utilizar las funciones más avanzadas cuando las necesites.

These days, the lines have blurred between text editors and IDEs. Most popular editors have some features that used to be exclusive to IDEs. Likewise, most IDEs can be configured to run in a lighter mode that's less distracting as you work, but lets you use the more advanced features when you need them.

Si ya tienes instalado un editor o IDE que te gusta, y si ya está configurado para funcionar con una versión reciente de Python instalada en tu sistema, te animo a que sigas con lo que ya conoces. Explorar diferentes editores puede ser divertido, pero también es una forma de evitar el trabajo de aprender un nuevo lenguaje.

If you already have an editor or IDE installed that you like, and if it's already configured to work with a recent version of Python that's installed on your system, then I encourage you to stick with what you already know. Exploring different editors can be fun, but it's also a way to avoid the work of learning a new language.

Si aún no tienes instalado un editor o IDE, te recomiendo VS Code por varias razones:

If you don't already have an editor or IDE installed, I recommend VS Code for a number of reasons:

Listado en español:

- Es gratuito y está publicado bajo una licencia de código abierto.
- Se puede instalar en los principales sistemas operativos.
- Es fácil de usar para principiantes, pero también lo suficientemente potente como para que muchos programadores profesionales lo utilicen como editor principal.
- Encuentra las versiones de Python que tienes instaladas, y normalmente no requiere ninguna configuración para ejecutar tus primeros programas.
- Tiene un terminal integrado, por lo que tu salida aparece en la misma ventana que tu código.
- Dispone de una extensión Python que hace que el editor sea muy eficaz para escribir y mantener código Python.
- Es altamente personalizable, por lo que puedes ajustarlo a tu forma de trabajar con el código.

Listing in English:

- It's free, and it's released under an open source license.
- It can be installed on all major operating systems.
- It's beginner-friendly but also powerful enough that many professional programmers use it as their main editor.
- It finds the versions of Python you have installed, and it typically does not require any configuration to run your first programs.
- It has an integrated terminal, so your output appears in the same window as your code.
- A Python extension is available that makes the editor highly efficient for writing and maintaining Python code.
- It's highly customizable, so you can tune it to match the way you work with code.

En este apéndice, aprenderás cómo empezar a configurar VS Code para que te funcione bien. También aprenderás algunos atajos que te permitirán trabajar de forma más eficiente.

Ser un mecanógrafo rápido no es tan importante como mucha gente piensa en programación, pero entender tu editor y saber cómo utilizarlo de forma eficiente es bastante útil.

In this appendix, you'll learn how to start configuring VS Code so that it works well for you. You'll also learn some shortcuts that let you work more efficiently. Being a fast typist is not as important as many people think in programming, but understanding your editor and knowing how to use it efficiently is quite helpful.

Dicho todo esto, VS Code no funciona para todo el mundo. Si por alguna razón no funciona bien en tu sistema, o si te distrae mientras trabajas, hay otros editores que pueden resultarte más atractivos. Este apéndice incluye una breve descripción de algunos de los otros editores e IDEs que deberías tener en cuenta.

With all that said, VS Code doesn't work for everyone. If it doesn't work well on your system for some reason, or if it's distracting you as you work, there are a number of other editors that you might find more appealing. This appendix includes a brief description of some of the other editors and IDEs you should consider.

Trabajar eficazmente con VS Code *(Working Efficiently with VS Code)*

En el Capítulo 1 , instalaste VS Code y añadiste también la extensión Python. Esta sección te mostrará algunas configuraciones adicionales que puedes hacer, además de atajos para trabajar eficientemente con tu código.

In Chapter 1 , you installed VS Code and added the Python extension as well. This section will show you some further configurations you can make, plus shortcuts for working efficiently with your code.

Configurar VS Code *(Configuring VS Code)*

Hay algunas formas de cambiar los ajustes de configuración por defecto de VS Code. Algunos cambios pueden hacerse a través de la interfaz, y otros requerirán cambios en los archivos de configuración. Estos cambios a veces tendrán efecto para todo lo que hagas en VS Code, mientras que otros sólo afectarán a los archivos dentro de la carpeta que contiene el archivo de configuración.

There are a few ways to change the default configuration settings for VS Code. Some changes can be made through the interface, and some will require changes in configuration files. These changes will sometimes take effect for everything you do in VS Code, while others will affect only the files within the folder that contains the configuration file.

Por ejemplo, si tienes un archivo de configuración en tu carpeta *python_work*, esos ajustes sólo afectarán a los archivos de esa carpeta (y sus subcarpetas). Esta es una buena característica, porque significa que puedes tener ajustes específicos del proyecto que anulen tus ajustes globales.

For example, if you have a configuration file in your *python_work* folder, those settings will only affect the files in that folder (and its subfolders). This is a good feature, because it means you can have project-specific settings that override your global settings.

Uso de tabulaciones y espacios (Using Tabs and Spaces)

Si utilizas una mezcla de tabuladores y espacios en tu código, puede causar problemas en tus programas que son difíciles de diagnosticar. Cuando trabajas en un archivo *.py* con la extensión Python instalada, VS Code está configurado para insertar cuatro espacios cada vez que pulsas la tecla TAB. Si sólo escribes tu propio código y tienes instalada la extensión Python, es probable que nunca tengas problemas con los tabuladores y los espacios.

If you use a mix of tabs and spaces in your code, it can cause problems in your programs that are difficult to diagnose. When working in a *.py* file with the Python extension installed, VS Code is configured to insert four spaces whenever you press the TAB key. If you're writing only your own code and you have the Python extension installed, you'll likely never have an issue with tabs and spaces.

Sin embargo, puede que tu instalación de VS Code no esté configurada correctamente. Además, en algún momento, puedes acabar trabajando en un archivo que sólo tenga tabuladores o una mezcla de tabuladores y espacios. Si sospechas que hay algún problema con las pestañas y los espacios, mira la barra de estado en la parte inferior de la ventana de VS Code y haz clic en **Espacios** o en **Tamaño de las pestañas**. Aparecerá un menú desplegable que te permite cambiar entre usar tabuladores y usar espacios. También puedes cambiar el nivel de sangría por defecto y convertir toda la sangría del archivo en

tabuladores o espacios.

However, your installation of VS Code may not be configured correctly. Also, at some point, you may end up working on a file that has only tabs or a mix of tabs and spaces. If you suspect any issue with tabs and spaces, look at the status bar at the bottom of the VS Code window and click either **Spaces** or **Tab Size** . A drop-down menu will appear that lets you switch between using tabs and using spaces. You can also change the default indentation level and convert all indentation in the file to either tabs or spaces.

Si estás mirando código y no estás seguro de si la sangría consiste en tabuladores o espacios, resalta varias líneas de código. Esto hará visibles los caracteres invisibles de los espacios en blanco. Cada espacio aparecerá como un punto, y cada tabulación como una flecha.

If you're looking at some code and you're not sure whether the indentation consists of tabs or spaces, highlight several lines of code. This will make the invisible whitespace characters visible. Each space will show up as a dot, and each tab will show up as an arrow.

Nota (Note)

En programación, se prefieren los espacios a las tabulaciones porque los espacios pueden ser interpretados sin ambigüedad por todas las herramientas que trabajan con un archivo de código. El ancho de los tabuladores puede ser interpretado de forma diferente por las distintas herramientas, lo que da lugar a errores que pueden ser extremadamente difíciles de diagnosticar.

In programming, spaces are preferred over tabs because spaces can be interpreted unambiguously by all tools that work with a code file. The width of tabs can be interpreted differently by different tools, which leads to errors that can be extremely difficult to diagnose.

Cambiar el tema de color (Changing the Color Theme)

VS Code utiliza un tema oscuro por defecto. Si quieres cambiarlo, haz clic en **Archivo** (**Código** en la barra de menús en macOS), luego en **Preferencias** y elige **Tema de color** . Aparecerá una lista desplegable que te permitirá elegir un tema que te vaya bien.

VS Code uses a dark theme by default. If you want to change this, click **File** (**Code** in the menu bar on macOS), then click **Preferences** and choose **Color Theme** . A drop-down list will appear, and it will let you choose a theme that works well for you.

Configurar el indicador de longitud de línea (Setting the Line Length Indicator)

La mayoría de los editores te permiten establecer un indicador visual, normalmente una línea vertical, para mostrar dónde deben terminar tus líneas. En la comunidad Python, la convención es limitar las líneas a 79 caracteres o menos.

Most editors allow you to set up a visual cue, usually a vertical line, to show where your lines should end. In the Python community, the convention is to limit lines to 79 characters or less.

Para configurar esta función, haz clic en **Código** y luego en **Preferencias** , y elige **Configuración** . En el cuadro de diálogo que aparece, introduce `rulers` . Verás una opción para `Editor: Reglas`; haz clic en el enlace etiquetado *Edit in settings.json* . En el archivo que aparece, añade lo siguiente a la configuración de `editor.rulers` :

To set this feature, click **Code** and then **Preferences** , and then choose **Settings** . In the dialog that appears, enter `rulers` . You'll see a setting for `Editor: Rulers`; click the link labeled *Edit in settings.json* . In the file that appears, add the following to the `editor.rulers` setting:

settings.json

settings.json

```
"editor.rulers": [  
  80,  
]
```

Esto añadirá una línea vertical en la ventana de edición en la posición de 80 caracteres. Puedes tener más de una línea vertical; por ejemplo, si quieres una línea adicional a 120 caracteres, el valor de tu configuración sería `[80, 120]` . Si no ves las líneas verticales, asegúrate de haber guardado el archivo de configuración; también es posible que tengas que salir y volver a abrir VS Code para que los cambios surtan efecto en algunos sistemas.

This will add a vertical line in the editing window at the 80-character position. You can have more than one vertical line; for example, if you want an additional line at 120 characters, the value for your setting would be `[80, 120]` . If you don't see the vertical lines, make sure you saved the settings file; you may also need to quit and reopen VS Code for the changes to take effect on some systems.

Simplificar la salida (Simplifying the Output)

Por defecto, VS Code muestra la salida de tus programas en una ventana de terminal incrustada. Esta salida incluye los comandos que se están utilizando para ejecutar el archivo. Para muchas situaciones, esto es ideal, pero puede distraerte más de lo que deseas cuando estás aprendiendo Python por primera vez.

By default, VS Code shows the output of your programs in an embedded terminal window. This output includes the commands that are being used to run the file. For many situations, this is ideal, but it might be more distracting than you want when you're first learning Python.

Para simplificar la salida, cierra todas las pestañas que estén abiertas en VS Code y luego sal de VS Code. Inicia de nuevo VS Code y abre la carpeta que contiene los archivos Python en los que estás trabajando; puede ser simplemente la carpeta *python_work* donde se guarda *hello_world.py* .

To simplify the output, close all the tabs that are open in VS Code and then quit VS Code. Launch VS Code again and open the folder that contains the Python files you're working on; this could just be the *python_work* folder where *hello_world.py* is saved.

Haz clic en el icono Ejecutar/Depurar (que parece un triángulo con un pequeño error) y, a continuación, haz clic en **Crear un archivo *launch.json*** . Selecciona las opciones de Python en las indicaciones que aparecen. En el archivo *launch.json* que se abre, realiza el siguiente cambio:

Click the Run/Debug icon (which looks like a triangle with a small bug), and then click **Create a *launch.json* File** . Select the Python options in the prompts that appear. In the *launch.json* file that opens, make the following change:

lanzamiento.json

launch.json

```
{
  --snip--
  "configurations": [
    {
      --snip--
      "console": "internalConsole",
      "justMyCode": true
    }
  ]
}
```

Aquí, estamos cambiando la configuración de `console` de `integratedTerminal` a `internalConsole` . Después de guardar el archivo de configuración, abre un archivo *.py* como *hello_world.py* , y ejecútalo pulsando CTRL-F5. En el panel de salida de VS Code, haz clic en **Consola de depuración** si no está ya seleccionada. Sólo deberías ver la salida de tu programa, y la salida debería actualizarse cada vez que ejecutes un programa.

Here, we're changing the `console` setting from `integratedTerminal` to `internalConsole` . After saving the settings file, open a *.py* file such as *hello_world.py* , and run it by pressing CTRL-F5. In the output pane of VS Code, click **Debug Console** if it's not already selected. You should see only your program's output, and the output should be refreshed every time you run a program.

Nota (Note)

La Consola de depuración es de sólo lectura. No funcionará con archivos que utilicen la función `input()` , que empezarás a utilizar en el Capítulo 7 . Cuando necesites ejecutar estos programas, puedes volver a cambiar la configuración de `console` a la predeterminada `integratedTerminal` , o puedes ejecutar estos programas en una ventana de terminal independiente, como se describe en " Ejecutar programas Python desde un terminal " en la página 11 .

The Debug Console is read-only. It won't work for files that use the `input()` function, which you'll start using in Chapter 7 . When you need to run these programs, you can either change the `console` setting back to the default `integratedTerminal` , or you can run these programs in a separate terminal window as described in " Running Python Programs from a Terminal " on page 11 .

Explorar otras personalizaciones (Exploring Further Customizations)

Puedes personalizar VS Code de muchas maneras para ayudarte a trabajar de forma más eficiente. Para empezar a explorar las personalizaciones disponibles, haz clic en **Código** y luego en **Preferencias** y, a continuación, elige **Configuración** . Verás una lista titulada Usos comunes; haz clic en cualquiera de los subtítulos para ver algunas formas comunes de modificar tu instalación de VS Code. Tómate tu tiempo para ver si hay alguna que haga que VS Code funcione mejor para ti, ¡pero no te pierdas tanto en la configuración de tu editor que dejes para más tarde el aprendizaje del uso de Python!

You can customize VS Code in many ways to help you work more efficiently. To start exploring the customizations available, click **Code** and then **Preferences** , and then choose **Settings** . You'll see a list titled Commonly Used; click any of the subheadings to see some common ways you can modify your installation of VS Code. Take some time to see if there are any that make VS Code work better for you, but don't get so lost in configuring your editor that you put off learning how to use Python!

Atajos de VS Code (VS Code Shortcuts)

Todos los editores e IDEs ofrecen formas eficientes de realizar tareas comunes que todo el mundo necesita hacer al escribir y mantener código. Por ejemplo, puedes sangrar fácilmente una sola línea de código o un bloque entero de código; con la misma facilidad

puedes mover un bloque de líneas hacia arriba o hacia abajo en un archivo.

All editors and IDEs offer efficient ways to do common tasks that everyone needs to do when writing and maintaining code. For example, you can easily indent a single line of code or an entire block of code; you can just as easily move a block of lines up or down in a file.

Hay demasiados atajos para describirlos aquí en su totalidad. Esta sección compartirá sólo algunos que probablemente te resultarán útiles cuando escribas tus primeros archivos Python. Si acabas utilizando un editor distinto de VS Code, asegúrate de aprender a realizar estas mismas tareas de forma eficaz en el editor que hayas elegido.

There are too many shortcuts to describe fully here. This section will share just a few that you'll likely find helpful as you're writing your first Python files. If you end up using a different editor than VS Code, make sure you learn how to do these same tasks efficiently in the editor you've chosen.

Sangrar y Desangrar Bloques de Código (Indenting and Unindenting Code Blocks)

Para sangrar un bloque entero de código, resáltalo y pulsa CTRL-], o ⌘-] en macOS. Para eliminar la sangría de un bloque de código, resáltalo y pulsa CTRL-[, o ⌘-[en macOS.

To indent an entire block of code, highlight it and press CTRL-], or ⌘-] on macOS. To unindent a block of code, highlight it and press CTRL-[, or ⌘-[on macOS.

Comentar bloques de código (Commenting Out Blocks of Code)

Para desactivar temporalmente un bloque de código, puedes resaltarlo y comentarlo para que Python lo ignore. Resalta la sección de código que quieras ignorar y pulsa CTRL-/, o ⌘- / en macOS. Las líneas seleccionadas se comentarán con una almohadilla (#) sangrada al mismo nivel que la línea de código, para indicar que no son comentarios normales. Cuando quieras descomentar el bloque de código, resalta el bloque y vuelve a ejecutar el mismo comando.

To temporarily disable a block of code, you can highlight the block and comment it so Python will ignore it. Highlight the section of code you want to ignore and press CTRL-/, or ⌘- / on macOS. The selected lines will be commented out with a hash mark (#) indented

at the same level as the line of code, to indicate these are not regular comments. When you want to uncomment the block of code, highlight the block and reissue the same command.

Subir o bajar líneas (Moving Lines Up or Down)

A medida que tus programas se hacen más complejos, puede que quieras mover un bloque de código hacia arriba o hacia abajo dentro de un archivo. Para ello, resalta el código que quieras mover y pulsa ALT-flecha arriba, u Opción-flecha arriba en macOS. La misma combinación de teclas con la flecha hacia abajo moverá el bloque hacia abajo en el archivo.

As your programs grow more complex, you may find that you want to move a block of code up or down within a file. To do so, highlight the code you want to move and press ALT-up arrow, or Option-up arrow on macOS. The same key combination with the down arrow will move the block down in the file.

Si vas a mover una sola línea hacia arriba o hacia abajo, puedes pulsar en cualquier lugar de esa línea; no necesitas resaltar toda la línea para moverla.

If you're moving a single line up or down, you can click anywhere in that line; you don't need to highlight the whole line to move it.

Ocultar el explorador de archivos (Hiding the File Explorer)

El explorador de archivos integrado en VS Code es realmente cómodo. Sin embargo, puede distraerte cuando estás escribiendo código y puede ocupar un espacio valioso en una pantalla más pequeña. El comando CTRL-B, o ⌘-B en macOS, alterna la visibilidad del panel del explorador de archivos.

The integrated file explorer in VS Code is really convenient. However, it can be distracting when you're writing code and can take up valuable space on a smaller screen. The command CTRL-B, or ⌘-B on macOS, toggles the visibility of the file explorer pane.

Encontrar atajos adicionales (Finding Additional Shortcuts)

Trabajar eficazmente en un entorno de edición requiere práctica, pero también intención. Cuando estés aprendiendo a trabajar con código, intenta fijarte en las cosas que haces

repetidamente. Cualquier acción que realices en tu editor probablemente tenga un atajo; si haces clic en elementos del menú para realizar tareas de edición, busca los atajos para esas acciones. Si cambias con frecuencia entre el teclado y el ratón, busca los atajos de navegación que evitan que tengas que coger el ratón tan a menudo.

Working efficiently in an editing environment takes practice, but it also takes intention. When you're learning to work with code, try to notice the things you do repeatedly. Any action you take in your editor likely has a shortcut; if you're clicking menu items to carry out editing tasks, look for the shortcuts for those actions. If you're switching between your keyboard and mouse frequently, look for the navigation shortcuts that keep you from reaching for your mouse so often.

Puedes ver todos los atajos de teclado en VS Code haciendo clic en **Código** y luego en **Preferencias** y, a continuación, seleccionando **Atajos de teclado**. Puedes utilizar la barra de búsqueda para encontrar un atajo concreto, o puedes desplazarte por la lista para encontrar atajos que te ayuden a trabajar de forma más eficiente.

You can see all the keyboard shortcuts in VS Code by clicking **Code** and then **Preferences**, and then choosing **Keyboard Shortcuts**. You can use the search bar to find a particular shortcut, or you can scroll through the list to find shortcuts that might help you work more efficiently.

Recuerda que es mejor centrarse en el código en el que estás trabajando y evitar dedicar demasiado tiempo a las herramientas que utilizas.

Remember, it's better to focus on the code that you're working on, and avoid spending too much time on the tools you're using.

Otros editores de texto e IDEs *(Other Text Editors and IDEs)*

Oirás hablar y verás a gente utilizando otros editores de texto. La mayoría de ellos pueden configurarse para ayudarte del mismo modo que has personalizado VS Code. He aquí una pequeña selección de editores de texto de los que puedes oír hablar.

You'll hear about and see people using a number of other text editors. Most of them can be configured to help you in the same way you've customized VS Code. Here's a small selection of text editors you might hear about.

IDLE (IDLE)

IDLE es un editor de texto que se incluye con Python. Es un poco menos intuitivo para trabajar que otros editores más modernos. Sin embargo, verás referencias a él en otros tutoriales dirigidos a principiantes, así que quizá quieras probarlo.

IDLE is a text editor that's included with Python. It's a little less intuitive to work with than other, more modern editors. However, you'll see references to it in other tutorials aimed at beginners, so you might want to give it a try.

Geany (Geany)

Geany es un sencillo editor de texto que muestra toda tu salida en una ventana de terminal separada, lo que te ayuda a sentirte cómodo utilizando terminales. Geany tiene una interfaz muy minimalista, pero es lo suficientemente potente como para que un número significativo de programadores experimentados sigan utilizándolo.

Geany is a simple text editor that displays all of your output in a separate terminal window, which helps you become comfortable using terminals. Geany has a very minimalist interface, but it's powerful enough that a significant number of experienced programmers still use it.

Si VS Code te distrae demasiado y está lleno de demasiadas funciones, considera la posibilidad de utilizar Geany en su lugar.

If you find VS Code too distracting and full of too many features, consider using Geany instead.

Sublime Text (Sublime Text)

Sublime Text es otro editor minimalista que deberías considerar utilizar si VS Code te parece demasiado recargado. Sublime Text tiene una interfaz realmente limpia y es

conocido por funcionar bien incluso con archivos muy grandes. Es un editor que se apartará de tu camino y te permitirá centrarte en el código que estás escribiendo.

Sublime Text is another minimalist editor that you should consider using if you find VS Code too busy. Sublime Text has a really clean interface and is known for working well even on very large files. It's an editor that will get out of your way and let you focus on the code you're writing.

Sublime Text tiene una prueba gratuita ilimitada, pero no es gratuito ni de código abierto. Si decides que te gusta y puedes permitirte comprar una licencia completa, deberías hacerlo. La compra es un pago único; no es una suscripción de software.

Sublime Text has an unlimited free trial, but it's not free or open source. If you decide you like it and can afford to purchase a full license, you should do so. The purchase is a one-time fee; it's not a software subscription.

Emacs y Vim (*Emacs and Vim*)

Emacs y *Vim* son dos editores populares preferidos por muchos programadores experimentados, porque están diseñados para que puedas utilizarlos sin que tus manos tengan que abandonar nunca el teclado. Esto hace que escribir, leer y modificar código sea muy eficaz, una vez que aprendes cómo funciona el editor. También significa que ambos editores tienen una curva de aprendizaje bastante pronunciada. Vim está incluido en la mayoría de las máquinas Linux y macOS, y tanto Emacs como Vim pueden ejecutarse completamente dentro de un terminal. Por esta razón, a menudo se utilizan para escribir código en servidores a través de sesiones de terminal remotas.

Emacs and *Vim* are two popular editors favored by many experienced programmers, because they're designed so you can use them without your hands ever having to leave the keyboard. This makes writing, reading, and modifying code very efficient, once you learn how the editor works. It also means both editors have a fairly steep learning curve. Vim is included on most Linux and macOS machines, and both Emacs and Vim can be run entirely inside a terminal. For this reason, they're often used to write code on servers through remote terminal sessions.

Los programadores suelen recomendarte que los pruebes, pero muchos programadores expertos olvidan lo mucho que los nuevos programadores ya están intentando aprender. Es

bueno que conozcas estos editores, pero deberías dejar de utilizarlos hasta que te sientas cómodo trabajando con código en un editor más fácil de usar que te permita centrarte en aprender a programar, en lugar de aprender a utilizar un editor.

Programmers will often recommend that you give them a try, but many proficient programmers forget how much new programmers are already trying to learn. It's good to be aware of these editors, but you should hold off on using them until you're comfortable working with code in a more user-friendly editor that lets you focus on learning to program, rather than learning to use an editor.

PyCharm (*PyCharm*)

PyCharm es un IDE popular entre los programadores de Python porque se creó para trabajar específicamente con Python. La versión completa requiere una suscripción de pago, pero también está disponible una versión gratuita llamada *PyCharm Community Edition*, y muchos desarrolladores la encuentran útil.

PyCharm is a popular IDE among Python programmers because it was built to work specifically with Python. The full version requires a paid subscription, but a free version called the *PyCharm Community Edition* is also available, and many developers find it useful.

Si pruebas *PyCharm*, ten en cuenta que, por defecto, configura un entorno aislado para cada uno de tus proyectos. Esto suele ser bueno, pero puede dar lugar a comportamientos inesperados si no entiendes lo que está haciendo por ti.

If you try *PyCharm*, be aware that, by default, it sets up an isolated environment for each of your projects. This is usually a good thing, but it can lead to unexpected behavior if you don't understand what it's doing for you.

Cuadernos Jupyter (*Jupyter Notebooks*)

Jupyter Notebook es un tipo de herramienta diferente de los editores de texto o IDE tradicionales, en el sentido de que es una aplicación web construida principalmente con bloques; cada bloque es un bloque de código o un bloque de texto. Los bloques de texto se representan en Markdown, por lo que puedes incluir un formato sencillo en tus bloques de texto.

Jupyter Notebook is a different kind of tool than traditional text editors or IDEs, in that it's a web app primarily built of blocks; each block is either a code block or a text block. The text blocks are rendered in Markdown, so you can include simple formatting in your text blocks.

Los Cuadernos Jupyter se desarrollaron para apoyar el uso de Python en aplicaciones científicas, pero desde entonces se han ampliado para ser útiles en una amplia variedad de situaciones. En lugar de limitarte a escribir comentarios dentro de un archivo `.py`, puedes escribir texto claro con un formato sencillo, como encabezados, listas con viñetas e hipervínculos entre secciones de código. Cada bloque de código puede ejecutarse independientemente, lo que te permite probar pequeñas partes de tu programa, o puedes ejecutar todos los bloques de código a la vez. Cada bloque de código tiene su propia área de salida, y puedes activar o desactivar las áreas de salida según necesites.

Jupyter Notebooks were developed to support the use of Python in scientific applications, but they have since expanded to become useful in a wide variety of situations. Rather than just writing comments inside a `.py` file, you can write clear text with simple formatting, such as headers, bulleted lists, and hyperlinks in between sections of code. Every code block can be run independently, allowing you to test small pieces of your program, or you can run all the code blocks at once. Each code block has its own output area, and you can toggle the output areas on or off as needed.

Los Cuadernos Jupyter pueden resultar confusos a veces debido a las interacciones entre las distintas celdas. Si defines una función en una celda, esa función está disponible también para otras celdas. Esto es beneficioso la mayor parte del tiempo, pero puede resultar confuso en cuadernos más largos y si no entiendes bien cómo funciona el entorno de los Cuadernos.

Jupyter Notebooks can be confusing at times because of the interactions between different cells. If you define a function in one cell, that function is available to other cells as well. This is beneficial most of the time, but it can be confusing in longer notebooks and if you don't fully understand how the Notebook environment works.

Si realizas algún trabajo científico o centrado en datos en Python, es casi seguro que en algún momento verás Cuadernos Jupyter.

If you're doing any scientific or data-focused work in Python, you'll almost certainly see Jupyter Notebooks at some point.

Apéndice C: Cómo obtener ayuda (Appendix C: Getting Help)

C

Obtener ayuda (C Getting Help)



Todo el mundo se atasca en algún momento cuando está aprendiendo a programar. Por eso, una de las habilidades más importantes que hay que aprender como programador es cómo desatascarse eficazmente. Este apéndice describe varias formas de ayudarte a volver a ponerte en marcha cuando la programación se vuelve confusa.

Everyone gets stuck at some point when they're learning to program. So, one of the most important skills to learn as a programmer is how to get unstuck efficiently. This appendix outlines several ways to help you get going again when programming gets confusing.

Primeros pasos (*First Steps*)

Cuando estés atascado, tu primer paso debe ser evaluar tu situación. Antes de pedir ayuda a nadie, responde con claridad a las tres preguntas siguientes:

When you're stuck, your first step should be to assess your situation. Before you ask for help from anyone else, answer the following three questions clearly:

Listado en español:

- ¿Qué intentas hacer?
- ¿Qué has intentado hasta ahora?
- ¿Qué resultados has obtenido?

Listing in English:

- What are you trying to do?
- What have you tried so far?
- What results have you been getting?

Haz que tus respuestas sean lo más concretas posible. Para la primera pregunta, afirmaciones explícitas como "Estoy intentando instalar la última versión de Python en mi nuevo portátil con Windows" son lo suficientemente detalladas como para que otros miembros de la comunidad Python puedan ayudarte. Afirmaciones como "Estoy intentando instalar Python" no proporcionan suficiente información para que otros puedan ofrecer mucha ayuda.

Make your answers as specific as possible. For the first question, explicit statements like "I'm trying to install the latest version of Python on my new Windows laptop" are detailed enough for others in the Python community to help you. Statements like "I'm trying to install Python" don't provide enough information for others to offer much help.

Tu respuesta a la segunda pregunta debe proporcionar suficientes detalles para que no te aconsejen repetir lo que ya has intentado: "Fui a <https://python.org/downloads> y pulsé el botón Descargar para mi sistema. Luego ejecuté el instalador" es más útil que "Fui al sitio web de Python y descargué algo".

Your answer to the second question should provide enough detail so you won't be advised to repeat what you've already tried: "I went to <https://python.org/downloads> and clicked the Download button for my system. Then I ran the installer" is more helpful than "I went to the Python website and downloaded something."

Para la tercera pregunta, es útil conocer los mensajes de error exactos que recibiste, de modo que puedas utilizarlos para buscar una solución en Internet o proporcionarlos cuando pidas ayuda.

For the third question, it's helpful to know the exact error messages you received, so you can use them to search online for a solution or provide them when asking for help.

A veces, el mero hecho de responder a estas tres preguntas antes de pedir ayuda a otros te permite ver algo que te falta, y te ayuda a desatascarte sin tener que ir más lejos. Los programadores incluso tienen un nombre para esto: *rubber duck debugging*. La idea es que si explicas claramente tu situación a un patito de goma (o a cualquier objeto inanimado) y le haces una pregunta concreta, a menudo serás capaz de responder a tu propia pregunta. Algunos equipos de programación incluso tienen cerca un pato de goma de verdad para animar a la gente a "hablar con el pato".

Sometimes, just answering these three questions before you ask for help from others allows you to see something you're missing, and helps get you unstuck without having to go any further. Programmers even have a name for this: *rubber duck debugging*. The idea is that if you clearly explain your situation to a rubber duck (or any inanimate object) and ask it a specific question, you'll often be able to answer your own question. Some programming teams even keep a real rubber duck around to encourage people to "talk to the duck."

Inténtalo de nuevo (*Try It Again*)

Volver al principio e intentarlo de nuevo puede ser suficiente para resolver muchos problemas. Digamos que estás intentando escribir un bucle `for` basándote en un ejemplo de este libro. Puede que sólo hayas pasado por alto algo sencillo, como dos puntos al final de la línea `for`. Repasar los pasos de nuevo podría ayudarte a evitar repetir el mismo error.

Just going back to the start and trying again can be enough to solve many problems. Say you're trying to write a `for` loop based on an example in this book. You might have only missed something simple, like a colon at the end of the `for` line. Going through the steps again might help you avoid repeating the same mistake.

Tómate un descanso (*Take a Break*)

Si llevas tiempo trabajando en el mismo problema, tomarte un descanso es una de las mejores tácticas que puedes probar. Cuando trabajamos en la misma tarea durante largos periodos de tiempo, nuestro cerebro empieza a centrarse en una única solución. Perdemos de vista las suposiciones que hemos hecho, y hacer una pausa nos ayuda a tener una nueva perspectiva del problema. No hace falta que sea una pausa larga, sólo algo que te saque de tu mentalidad actual. Si llevas mucho tiempo sentado, haz algo físico: da un breve paseo, sal un rato al exterior, o tal vez bebe un vaso de agua o come un tentempié ligero.

If you've been working on the same problem for a while, taking a break is one of the best tactics you can try. When we work on the same task for long periods of time, our brains start to zero in on only one solution. We lose sight of the assumptions we've made, and taking a break helps us get a fresh perspective on the problem. It doesn't need to be a long break, just something that gets you out of your current mindset. If you've been sitting for a long time, do something physical: take a short walk, go outside for a bit, or perhaps drink a glass of water or eat a light snack.

Si te sientes frustrado, quizá merezca la pena dejar el trabajo a un lado por hoy. Una buena noche de sueño casi siempre hace que un problema sea más abordable.

If you're getting frustrated, it might be worth putting your work away for the day. A good night's sleep almost always makes a problem more approachable.

Consulta los recursos de este libro (*Refer to This Book's Resources*)

Los recursos en línea de este libro, disponibles en https://ehmatthes.github.io/pcc_3e, incluyen una serie de secciones útiles sobre cómo configurar tu sistema y trabajar en cada capítulo. Si aún no lo has hecho, echa un vistazo a estos recursos y comprueba si hay algo que te ayude en tu situación.

The online resources for this book, available at https://ehmatthes.github.io/pcc_3e, include a number of helpful sections about setting up your system and working through each chapter. If you haven't done so already, take a look at these resources and see if there's anything that helps your situation.

Búsqueda en Internet (*Searching Online*)

Es muy probable que alguien haya tenido el mismo problema que tú y haya escrito sobre ello en Internet. Unas buenas habilidades de búsqueda e indagaciones específicas te ayudarán a encontrar recursos existentes para resolver el problema al que te enfrentas. Por ejemplo, si tienes problemas para instalar la última versión de Python en un nuevo sistema Windows, buscar *install python windows* y limitar los resultados a recursos del último año podría dirigirte a una respuesta clara.

Chances are good that someone else has had the same problem you're having and has written about it online. Good searching skills and specific inquiries will help you find existing resources to solve the issue you're facing. For example, if you're struggling to install the latest version of Python on a new Windows system, searching for *install python windows* and limiting the results to resources from the last year might direct you to a clear answer.

Buscar el mensaje de error exacto también puede ser muy útil. Por ejemplo, supongamos que recibes el siguiente error cuando intentas ejecutar un programa Python desde un terminal en un nuevo sistema Windows:

Searching the exact error message can be extremely helpful too. For example, say you get the following error when you try to run a Python program from a terminal on a new Windows system:

```
> python hello_world.py
Python was not found; run without arguments to install from the Microsoft
Store...
```

Buscar la frase completa, "No se encontró Python; ejecutar sin argumentos para instalar desde la Tienda Microsoft", probablemente te dará algún buen consejo.

Searching for the full phrase, "Python was not found; run without arguments to install from the Microsoft Store," will probably yield some good advice.

Cuando empieces a buscar temas relacionados con la programación, aparecerán repetidamente algunos sitios. Describiré brevemente algunos de estos sitios, para que sepas hasta qué punto pueden serte útiles.

When you start searching for programming-related topics, a few sites will appear repeatedly. I'll describe some of these sites briefly, so you'll know how helpful they're likely to be.

Stack Overflow (*Stack Overflow*)

Stack Overflow (<https://stackoverflow.com>) es uno de los sitios más populares de preguntas y respuestas para programadores, y a menudo aparecerá en la primera página de resultados de las búsquedas relacionadas con Python. Los miembros publican preguntas cuando están atascados, y otros miembros intentan dar respuestas útiles. Los usuarios pueden votar las respuestas que consideran más útiles, por lo que las mejores respuestas suelen ser las primeras que encontrarás.

Stack Overflow (<https://stackoverflow.com>) is one of the most popular question-and-answer sites for programmers, and it will often appear in the first page of results on Python-related searches. Members post questions when they're stuck, and other members try to give helpful responses. Users can vote for the responses they find most helpful, so the best answers are usually the first ones you'll find.

Muchas preguntas básicas sobre Python tienen respuestas muy claras en Stack Overflow, porque la comunidad las ha perfeccionado con el tiempo. También se anima a los usuarios a publicar actualizaciones, por lo que las respuestas tienden a mantenerse relativamente actualizadas. En el momento de escribir esto, se han respondido casi dos millones de preguntas relacionadas con Python en Stack Overflow.

Many basic Python questions have very clear answers on Stack Overflow, because the community has refined them over time. Users are encouraged to post updates, too, so responses tend to stay relatively current. At the time of this writing, almost two million Python-related questions have been answered on Stack Overflow.

Hay una expectativa que debes tener en cuenta antes de publicar en Stack Overflow. Las preguntas deben ser el ejemplo más breve del tipo de problema al que te enfrentas. Si publicas entre 5 y 20 líneas de código que generen el error al que te enfrentas, y si respondes a las preguntas mencionadas en " Primeros pasos " en la página 477 de este apéndice, es probable que alguien te ayude. Si compartes un enlace a un proyecto con varios archivos grandes, es muy poco probable que la gente te ayude. Hay una guía

estupenda para redactar una buena pregunta en <https://stackoverflow.com/help/how-to-ask> . Las sugerencias de esta guía son aplicables a la obtención de ayuda en cualquier comunidad de programadores.

There's one expectation you should be aware of before posting on Stack Overflow. Questions are meant to be the shortest example of the kind of issue you're facing. If you post 5–20 lines of code that generate the error you're facing, and if you address the questions mentioned in “ First Steps ” on page 477 earlier in this appendix, someone will probably help you. If you share a link to a project with multiple large files, people will be very unlikely to help. There's a great guide to writing up a good question at <https://stackoverflow.com/help/how-to-ask> . The suggestions in this guide are applicable to getting help in any community of programmers.

La documentación oficial de Python (*The Official Python Documentation*)

La documentación oficial de Python (<https://docs.python.org>) es un poco más imprecisa para los principiantes, porque su propósito es más documentar el lenguaje que proporcionar explicaciones. Los ejemplos de la documentación oficial deberían funcionar, pero es posible que no entiendas todo lo que se muestra. Aún así, es un buen recurso para consultar cuando aparezca en tus búsquedas, y te resultará más útil a medida que sigas ampliando tus conocimientos de Python.

The official Python documentation (<https://docs.python.org>) is a bit more hit-or-miss for beginners, because its purpose is more to document the language than to provide explanations. The examples in the official documentation should work, but you might not understand everything shown. Still, it's a good resource to check when it comes up in your searches, and it will become more useful to you as you continue building your understanding of Python.

Documentación oficial de bibliotecas (*Official Library Documentation*)

Si utilizas una biblioteca específica, como Pygame, Matplotlib o Django, los enlaces a la documentación oficial de la misma aparecerán a menudo en las búsquedas. Por ejemplo, <https://docs.djangoproject.com> es muy útil cuando se trabaja con Django. Si piensas trabajar con alguna de estas bibliotecas, es una buena idea que te familiarices con su documentación oficial.

If you're using a specific library, such as Pygame, Matplotlib, or Django, links to the official documentation for it will often appear in searches. For example,

<https://docs.djangoproject.com> is very helpful when working with Django. If you're planning to work with any of these libraries, it's a good idea to become familiar with their official documentation.

[r/learnpython](#) (*r/learnpython*)

Reddit se compone de una serie de subforos denominados *subreddits*. El subreddit *r/learnpython* (<https://reddit.com/r/learnpython>) es muy activo y solidario. Puedes leer las preguntas de los demás y publicar también las tuyas. A menudo obtendrás múltiples perspectivas sobre las preguntas que plantees, lo que puede ser realmente útil para comprender mejor el tema en el que estás trabajando.

Reddit is made up of a number of subforums called *subreddits*. The *r/learnpython* subreddit (<https://reddit.com/r/learnpython>) is very active and supportive. You can read others' questions and post your own as well. You will often get multiple perspectives about the questions you raise, which can be really helpful in gaining a deeper understanding of the topic you're working on.

Publicaciones en blogs (*Blog Posts*)

Muchos programadores mantienen blogs y comparten posts sobre las partes del lenguaje con las que están trabajando. Deberías buscar una fecha en las entradas de blog que encuentres, para ver hasta qué punto es probable que la información sea aplicable a la versión de Python que estás utilizando.

Many programmers maintain blogs and share posts about the parts of the language they're working with. You should look for a date on the blog posts you find, to see how applicable the information is likely to be for the version of Python you're using.

Discord (*Discord*)

Discord es un entorno de chat en línea con una comunidad Python donde puedes pedir ayuda y seguir discusiones relacionadas con Python.

Discord is an online chat environment with a Python community where you can ask for help and follow Python-related discussions.

Para comprobarlo, dirígete a <https://pythondiscord.com> y haz clic en el enlace **Discord** de la parte superior derecha. Si ya tienes una cuenta de Discord, puedes conectarte con ella. Si no tienes una cuenta, introduce un nombre de usuario y sigue las indicaciones para completar tu registro en Discord.

To check it out, head to <https://pythondiscord.com> and click the **Discord** link at the upper right. If you already have a Discord account, you can log in with your existing account. If you don't have an account, enter a username and follow the prompts to complete your Discord registration.

Si es la primera vez que visitas el Discord de Python, tendrás que aceptar las normas de la comunidad antes de participar plenamente. Una vez hecho esto, puedes unirte a cualquiera de los canales que te interesen. Si buscas ayuda, asegúrate de publicar en uno de los canales de Ayuda de Python.

If this is your first time visiting the Python Discord, you'll need to accept the rules for the community before participating fully. Once you've done that, you can join any of the channels that interest you. If you're looking for help, be sure to post in one of the Python Help channels.

Slack (Slack)

Slack es otro entorno de chat online. Suele utilizarse para las comunicaciones internas de la empresa, pero también hay muchos grupos públicos a los que puedes unirte. Si quieres echar un vistazo a los grupos Slack de Python, empieza por <https://pyslackers.com>. Haz clic en el enlace **Slack** de la parte superior de la página y, a continuación, introduce tu dirección de correo electrónico para obtener una invitación.

Slack is another online chat environment. It is often used for internal company communications, but there are also many public groups you can join. If you want to check out Python Slack groups, start with <https://pyslackers.com>. Click the **Slack** link at the top of the page, then enter your email address to get an invitation.

Una vez que estés en el espacio de trabajo Desarrolladores Python, verás una lista de canales. Haz clic en **Canales** y elige los temas que te interesen. Quizá quieras empezar por los canales *#help* y *#django*.

Once you're in the Python Developers workspace, you'll see a list of channels. Click **Channels** and then choose the topics that interest you. You might want to start with the *#help* and *#django* channels.

Apéndice D: Uso de Git para el control de versiones (Appendix D: Using Git for Version Control)

D

Utilizar Git para el control de versiones *(D Using Git for Version Control)*



El software de control de versiones te permite tomar instantáneas de un proyecto siempre que esté en estado de trabajo. Cuando realizas cambios en un proyecto -por ejemplo, cuando implementas una nueva función- puedes volver a un estado de trabajo anterior si el estado actual del proyecto no funciona bien.

Version control software allows you to take snapshots of a project whenever it's in a working state. When you make changes to a project—for example, when you implement a new feature—you can go back to a previous working state if the project's current state isn't functioning well.

Utilizar un software de control de versiones te da libertad para trabajar en mejoras y cometer errores sin preocuparte de arruinar el proyecto. Esto es especialmente crítico en proyectos grandes, pero también puede ser útil en proyectos más pequeños, incluso cuando trabajas con programas contenidos en un único archivo.

Using version control software gives you the freedom to work on improvements and make mistakes without worrying about ruining your project. This is especially critical in large projects, but can also be helpful in smaller projects, even when you're working on programs contained in a single file.

En este apéndice, aprenderás a instalar Git y a utilizarlo para el control de versiones en los programas en los que estés trabajando ahora. *Git* es el software de control de versiones más popular que se utiliza hoy en día. Muchas de sus herramientas avanzadas ayudan a los equipos a colaborar en grandes proyectos, pero sus funciones más básicas también funcionan bien para desarrolladores en solitario. Git implementa el control de versiones mediante el seguimiento de los cambios realizados en cada archivo de un proyecto; si cometes un error, sólo tienes que volver a un estado previamente guardado.

In this appendix, you'll learn to install Git and use it for version control in the programs you're working on now. *Git* is the most popular version control software in use today. Many of its advanced tools help teams collaborate on large projects, but its most basic features also work well for solo developers. Git implements version control by tracking the changes made to every file in a project; if you make a mistake, you can just return to a previously saved state.

Instalación de Git (*Installing Git*)

Git funciona en todos los sistemas operativos, pero hay diferentes enfoques para instalarlo en cada sistema. Las siguientes secciones proporcionan instrucciones específicas para cada sistema operativo.

Git runs on all operating systems, but there are different approaches to installing it on each system. The following sections provide specific instructions for each operating system.

Git está incluido por defecto en algunos sistemas, y a menudo se incluye con otros paquetes que puede que ya tengas instalados. Antes de intentar instalar Git, comprueba si ya está en tu sistema. Abre una nueva ventana de terminal y ejecuta el comando `git --version`. Si

aparece un número de versión específico, Git está instalado en tu sistema. Si ves un mensaje pidiéndote que instales o actualices Git, sigue las instrucciones en pantalla.

Git is included on some systems by default, and is often bundled with other packages that you might have already installed. Before trying to install Git, see if it's already on your system. Open a new terminal window and issue the command `git --version` . If you see output listing a specific version number, Git is installed on your system. If you see a message prompting you to install or update Git, follow the onscreen instructions.

Si no ves ninguna instrucción en pantalla y utilizas Windows o macOS, puedes descargar un instalador de <https://git-scm.com> . Si eres usuario de Linux con un sistema compatible con apt, puedes instalar Git con el comando `sudo apt install git` .

If you don't see any onscreen instructions and you're using Windows or macOS, you can download an installer from <https://git-scm.com> . If you're a Linux user with an apt-compatible system, you can install Git with the command `sudo apt install git` .

Configurar Git (Configuring Git)

Git realiza un seguimiento de quién realiza cambios en un proyecto, incluso cuando sólo una persona está trabajando en el proyecto. Para ello, Git necesita saber tu nombre de usuario y tu correo electrónico. Debes proporcionar un nombre de usuario, pero puedes inventarte una dirección de correo electrónico falsa:

Git keeps track of who makes changes to a project, even when only one person is working on the project. To do this, Git needs to know your username and email. You must provide a username, but you can make up a fake email address:

```
$ git config --global user.name "username"
$ git config --global user.email "username@example.com"
```

Si olvidas este paso, Git te pedirá esta información cuando hagas tu primera confirmación.

If you forget this step, Git will prompt you for this information when you make your first commit.

También es mejor establecer el nombre por defecto para la rama principal de cada proyecto. Un buen nombre para esta rama es `main` :

It's also best to set the default name for the main branch in each project. A good name for this branch is `main` :

```
$ git config --global init.defaultBranch main
```

Esta configuración significa que cada nuevo proyecto que gestiones con Git empezará con una única rama de confirmaciones llamada *main* .

This configuration means that each new project you use Git to manage will start out with a single branch of commits called *main* .

Crear un proyecto (*Making a Project*)

Vamos a crear un proyecto con el que trabajar. Crea una carpeta en algún lugar de tu sistema llamada *git_practice* . Dentro de la carpeta, crea un sencillo programa Python:

Let's make a project to work with. Create a folder somewhere on your system called *git_practice* . Inside the folder, make a simple Python program:

```
hola_git.py
```

```
hello_git.py
```

```
print("Hello Git world!")
```

Utilizaremos este programa para explorar la funcionalidad básica de Git.

We'll use this program to explore Git's basic functionality.

Ignorar archivos (*Ignoring Files*)

Los archivos con la extensión `.pyc` se generan automáticamente a partir de los archivos `.py`, por lo que no necesitamos que Git haga un seguimiento de ellos. Estos archivos se almacenan en un directorio llamado `__pycache__`. Para decirle a Git que ignore este directorio, crea un archivo especial llamado `.gitignore` -con un punto al principio del nombre del archivo y sin extensión- y añádele la siguiente línea

Files with the extension `.pyc` are automatically generated from `.py` files, so we don't need Git to keep track of them. These files are stored in a directory called `__pycache__`. To tell Git to ignore this directory, make a special file called `.gitignore` —with a dot at the beginning of the filename and no file extension—and add the following line to it:

`.gitignore`

`.gitignore`

`__pycache__/`

Este archivo indica a Git que ignore cualquier archivo del directorio `__pycache__`. Utilizar un archivo `.gitignore` mantendrá tu proyecto libre de desorden y facilitará el trabajo.

This file tells Git to ignore any file in the `__pycache__` directory. Using a `.gitignore` file will keep your project clutter-free and easier to work with.

Puede que tengas que modificar la configuración de tu explorador de archivos para que se muestren los archivos ocultos (archivos cuyos nombres empiezan por un punto). En el Explorador de Windows, marca la casilla del menú Ver denominada **Elementos ocultos**. En macOS, pulsa **⌘**-SHIFT-. (punto). En Linux, busca la opción Mostrar archivos ocultos.

You might need to modify your file browser's settings so hidden files (files whose names begin with a dot) will be shown. In Windows Explorer, check the box in the View menu labeled **Hidden Items**. On macOS, press **⌘**-SHIFT-. (dot). On Linux, look for a setting labeled Show Hidden Files.

Nota (Note)

Si estás en macOS, añade una línea más a *.gitignore* . Añade el nombre *.DS_Store* ; son archivos ocultos que contienen información sobre cada directorio en macOS, y desordenarán tu proyecto si no los añades a *.gitignore* .

If you're on macOS, add one more line to *.gitignore* . Add the name *.DS_Store* ; these are hidden files that contain information about each directory on macOS, and they will clutter up your project if you don't add them to *.gitignore* .

Inicializar un repositorio (*Initializing a Repository*)

Ahora que tienes un directorio que contiene un archivo Python y un archivo *.gitignore* , puedes inicializar un repositorio Git. Abre un terminal, navega hasta la carpeta *git_practice* y ejecuta el siguiente comando:

Now that you have a directory containing a Python file and a *.gitignore* file, you can initialize a Git repository. Open a terminal, navigate to the *git_practice* folder, and run the following command:

```
git_practice$ git init
Initialized empty Git repository in git_practice/.git/
git_practice$
```

La salida muestra que Git ha inicializado un repositorio vacío en *git_practice* . Un *repository* es el conjunto de archivos de un programa que Git está rastreando activamente. Todos los archivos que Git utiliza para gestionar el repositorio se encuentran en el directorio oculto *.git* , con el que no necesitarás trabajar en absoluto. Simplemente no borres ese directorio, o perderás el historial de tu proyecto.

The output shows that Git has initialized an empty repository in *git_practice* . A *repository* is the set of files in a program that Git is actively tracking. All the files Git uses to manage the

repository are located in the hidden directory *.git* , which you won't need to work with at all. Just don't delete that directory, or you'll lose your project's history.

Comprobar el estado (*Checking the Status*)

Antes de hacer nada más, echemos un vistazo al estado del proyecto:

Before doing anything else, let's look at the project's status:

```
git_practice$ git status
```

❶ On branch main

No commits yet

❷ Untracked files:

(use "git add <file>..." to include in what will be committed)

.gitignore

hello_git.py

❸ nothing added to commit but untracked files present (use "git add" to track)

```
git_practice$
```

En Git, una *branch* es una versión del proyecto en la que estás trabajando; aquí puedes ver que estamos en una rama llamada `main` ❶ . Cada vez que compruebes el estado de tu proyecto, debería mostrar que estás en la rama `main` . Entonces verás que estamos a punto de hacer el commit inicial. Un *commit* es una instantánea del proyecto en un momento determinado.

In Git, a *branch* is a version of the project you're working on; here you can see that we're on a branch named `main` ❶ . Each time you check your project's status, it should show that you're on the branch `main` . You then see that we're about to make the initial commit. A *commit* is a snapshot of the project at a particular point in time.

Git nos informa de que hay archivos sin rastrear en el proyecto ❷ , porque aún no le hemos dicho qué archivos rastrear. Luego nos dice que no se ha añadido nada a la confirmación actual, pero que hay archivos sin rastrear que quizá queramos añadir al repositorio ❸ .

Git informs us that untracked files are in the project ❷ , because we haven't told it which files to track yet. Then we're told that there's nothing added to the current commit, but untracked files are present that we might want to add to the repository ❸ .

Añadir archivos al repositorio (*Adding Files to the Repository*)

Añadamos los dos archivos al repositorio y comprobemos de nuevo el estado:

Let's add the two files to the repository and check the status again:

```
❶ git_practice$ git add .
```

```
❷ git_practice$ git status
```

```
On branch main
```

```
No commits yet
```

```
Changes to be committed:
```

```
(use "git rm --cached <file>..." to unstage)
```

```
❸ new file: .gitignore
```

```
new file: hello_git.py
```

```
git_practice$
```

El comando `git add .` añade al repositorio todos los archivos de un proyecto que aún no estén siendo rastreados ❶ , siempre que no estén listados en `.gitignore` . No confirma los archivos; sólo le dice a Git que empiece a prestarles atención. Cuando comprobamos ahora el estado del proyecto, podemos ver que Git reconoce algunos cambios que necesitan ser confirmados ❷ . La etiqueta *new file* significa que estos archivos se han añadido recientemente al repositorio ❸ .

The command `git add .` adds to the repository all files within a project that aren't already being tracked ❶ , as long as they're not listed in `.gitignore` . It doesn't commit the files; it just tells Git to start paying attention to them. When we check the status of the project now, we can see that Git recognizes some changes that need to be committed ❷ . The label *new file* means these files were newly added to the repository ❸ .

Hacer un commit *(Making a Commit)*

Hagamos el primer commit:

Let's make the first commit:

```
❶ git_practice$ git commit -m "Started project."
❷ [main (root-commit) cea13dd] Started project.
❸ 2 files changed, 5 insertions(+)
   create mode 100644 .gitignore
   create mode 100644 hello_git.py
❹ git_practice$ git status
On branch main
nothing to commit, working tree clean
git_practice$
```

Emitimos el comando `git commit -m " message "` ❶ para hacer una instantánea del proyecto. La bandera `-m` indica a Git que registre el mensaje que sigue (`Started proyecto.`) en el registro del proyecto. La salida muestra que estamos en la rama `main` ❷ y que dos archivos han cambiado ❸ .

We issue the command `git commit -m " message "` ❶ to make a snapshot of the project. The `-m` flag tells Git to record the message that follows (`Started project.`) in the project's log. The output shows that we're on the `main` branch ❷ and that two files have changed ❸ .

Si ahora comprobamos el estado, veremos que estamos en la rama `main` y que tenemos un árbol de trabajo limpio ❹ . Este es el mensaje que deberías ver cada vez que confirmes un estado de trabajo de tu proyecto. Si recibes un mensaje diferente, léelo atentamente; es probable que hayas olvidado añadir un archivo antes de hacer un commit.

When we check the status now, we can see that we're on the `main` branch, and we have a clean working tree ❹ . This is the message you should see each time you commit a working state of your project. If you get a different message, read it carefully; it's likely you forgot to add a file before making a commit.

Comprobación del registro (*Checking the Log*)

Git mantiene un registro de todas las confirmaciones realizadas en el proyecto. Vamos a comprobar el registro:

Git keeps a log of all commits made to the project. Let's check the log:

```
git_practice$ git log
commit cea13ddc51b885d05a410201a54faf20e0d2e246 (HEAD -> main)
Author: eric <eric@example.com>
Date: Mon Jun 6 19:37:26 2022 -0800
```

```
    Started project.
git_practice$
```

Cada vez que realizas una confirmación, Git genera un ID de referencia único de 40 caracteres. Registra quién hizo la confirmación, cuándo se hizo y el mensaje registrado. No siempre necesitarás toda esta información, por lo que Git proporciona una opción para imprimir una versión más simple de las entradas del registro:

Each time you make a commit, Git generates a unique, 40-character reference ID. It records who made the commit, when it was made, and the message recorded. You won't always need all of this information, so Git provides an option to print a simpler version of the log entries:

```
git_practice$ git log --pretty=oneline
cea13ddc51b885d05a410201a54faf20e0d2e246 (HEAD -> main) Started project.
git_practice$
```

La bandera `--pretty=oneline` proporciona las dos piezas de información más importantes: el ID de referencia de la confirmación y el mensaje registrado para la confirmación.

The `--pretty=oneline` flag provides the two most important pieces of information: the reference ID of the commit and the message recorded for the commit.

La segunda confirmación (*The Second Commit*)

Para ver el verdadero poder del control de versiones, necesitamos hacer un cambio en el proyecto y confirmar ese cambio. Aquí sólo añadiremos otra línea a *hello_git.py* :

To see the real power of version control, we need to make a change to the project and commit that change. Here we'll just add another line to *hello_git.py* :

hola_git.py

hello_git.py

```
print("Hello Git world!")  
print("Hello everyone.")
```

Cuando comprobemos el estado del proyecto, veremos que Git se ha dado cuenta del archivo que ha cambiado:

When we check the status of the project, we'll see that Git has noticed the file that changed:

```
git_practice$ git status
```

❶ On branch main

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

❷ modified: hello_git.py

❸ no changes added to commit (use "git add" and/or "git commit -a")

```
git_practice$
```

Vemos la rama en la que estamos trabajando ❶ , el nombre del archivo que se ha modificado ❷ , y que no se ha confirmado ningún cambio ❸ . Confirmemos el cambio y comprobemos de nuevo el estado:

We see the branch we're working on ❶ , the name of the file that was modified ❷ , and that no changes have been committed ❸ . Let's commit the change and check the status again:

```
❶ git_practice$ git commit -am "Extended greeting."
[main 945fa13] Extended greeting.
1 file changed, 1 insertion(+), 1 deletion(-)
❷ git_practice$ git status
On branch main
nothing to commit, working tree clean
❸ git_practice$ git log --pretty=oneline
945fa13af128a266d0114eebb7a3276f7d58ecd2 (HEAD -> main) Extended greeting.
cea13ddc51b885d05a410201a54faf20e0d2e246 Started project.
git_practice$
```

Hacemos un nuevo commit, pasando las banderas `-am` cuando utilizemos el comando `git commit` ❶ . La bandera `-a` indica a Git que añada todos los archivos modificados en el repositorio a la confirmación actual. (Si creas algún archivo nuevo entre confirmaciones, vuelve a emitir el comando `git add` . para incluir los nuevos archivos en el repositorio). La bandera `-m` indica a Git que registre un mensaje en el registro para esta confirmación.

We make a new commit, passing the `-am` flags when we use the command `git commit` ❶ . The `-a` flag tells Git to add all modified files in the repository to the current commit. (If you create any new files between commits, reissue the `git add` . command to include the new files in the repository.) The `-m` flag tells Git to record a message in the log for this commit.

Cuando comprobamos el estado del proyecto, vemos que de nuevo tenemos un árbol de trabajo limpio ❷ . Por último, vemos las dos confirmaciones en el registro ❸ .

When we check the project's status, we see that we once again have a clean working tree ❷ . Finally, we see the two commits in the log ❸ .

Abandonar cambios (*Abandoning Changes*)

Veamos ahora cómo abandonar un cambio y volver al estado de trabajo anterior. Primero, añade una nueva línea a `hello_git.py` :

Now let's look at how to abandon a change and go back to the previous working state. First, add a new line to *hello_git.py* :

hola_git.py

hello_git.py

```
print("Hello Git world!")  
print("Hello everyone.")
```

```
print("Oh no, I broke the project!")
```

Guarda y ejecuta este archivo.

Save and run this file.

Comprobamos el estado y vemos que Git se da cuenta de este cambio:

We check the status and see that Git notices this change:

```
git_practice$ git status
```

```
On branch main
```

```
Changes not staged for commit:
```

```
(use "git add <file>..." to update what will be committed)
```

```
(use "git restore <file>..." to discard changes in working directory)
```

```
❶  modified:  hello_git.py
```

```
no changes added to commit (use "git add" and/or "git commit -a")
```

```
git_practice$
```

Git ve que hemos modificado *hello_git.py* ❶ , y podemos confirmar el cambio si queremos. Pero esta vez, en lugar de confirmar el cambio, volveremos a la última confirmación, cuando sabíamos que nuestro proyecto funcionaba. No haremos nada en *hello_git.py* : no borraremos la línea ni utilizaremos la función Deshacer del editor de texto. En su lugar, introduce los siguientes comandos en tu sesión de terminal:

Git sees that we modified *hello_git.py* ❶ , and we can commit the change if we want to. But this time, instead of committing the change, we'll go back to the last commit when we knew our project was working. We won't do anything to *hello_git.py* : we won't delete the line or use the Undo feature in the text editor. Instead, enter the following commands in your terminal session:

```
git_practice$ git restore .
git_practice$ git status
On branch main
nothing to commit, working tree clean
git_practice$
```

El comando `git restore filename` te permite abandonar todos los cambios desde la última confirmación en un archivo específico. El comando `git restore .` abandona todos los cambios realizados en todos los archivos desde la última confirmación; esta acción restaura el proyecto al último estado confirmado.

The command `git restore filename` allows you to abandon all changes since the last commit in a specific file. The command `git restore .` abandons all changes made in all files since the last commit; this action restores the project to the last committed state.

Cuando vuelvas a tu editor de texto, verás que *hello_git.py* ha vuelto a este estado:

When you return to your text editor, you'll see that *hello_git.py* has changed back to this:

```
print("Hello Git world!")
print("Hello everyone.")
```

Aunque volver a un estado anterior pueda parecer trivial en este sencillo proyecto, si estuviéramos trabajando en un proyecto grande con docenas de archivos modificados, se

restaurarían todos los archivos que hubieran cambiado desde el último commit. Esta función es increíblemente útil: puedes hacer tantos cambios como quieras al implementar una nueva función, y si no funcionan, puedes descartarlos sin afectar al proyecto. No tienes que recordar esos cambios y deshacerlos manualmente. Git hace todo eso por ti.

Although going back to a previous state might seem trivial in this simple project, if we were working on a large project with dozens of modified files, all the files that had changed since the last commit would be restored. This feature is incredibly useful: you can make as many changes as you want when implementing a new feature, and if they don't work, you can discard them without affecting the project. You don't have to remember those changes and manually undo them. Git does all of that for you.

Nota (Note)

Puede que tengas que actualizar el archivo en tu editor para ver la versión restaurada.

You might have to refresh the file in your editor to see the restored version.

Revisar confirmaciones anteriores (Checking Out Previous Commits)

Puedes revisar cualquier confirmación en tu registro, utilizando el comando `checkout`, utilizando los seis primeros caracteres de un ID de referencia. Tras comprobar y revisar una confirmación anterior, puedes volver a la última confirmación o abandonar tu trabajo reciente y retomar el desarrollo a partir de la confirmación anterior:

You can revisit any commit in your log, using the `checkout` command, by using the first six characters of a reference ID. After checking out and reviewing an earlier commit, you can return to the latest commit or abandon your recent work and pick up development from the earlier commit:

```
git_practice$ git log --pretty=oneline
945fa13af128a266d0114eebb7a3276f7d58ecd2 (HEAD -> main) Extended greeting.
cea13ddc51b885d05a410201a54faf20e0d2e246 Started project.
```

```
git_practice$ git checkout cea13d
Note: switching to 'cea13d'.
```

❶ You are in 'detached HEAD' state. You can look around, make experimental changes and commit them, and you can discard any commits you make in this state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may do so (now or later) by using -c with the switch command. Example:

```
git switch -c <new-branch-name>
```

❷ Or undo this operation with:

```
git switch -
```

Turn off this advice by setting config variable advice.detachedHead to false

```
HEAD is now at cea13d Started project.
git_practice$
```

Cuando revisas una confirmación anterior, abandonas la rama principal y entras en lo que Git denomina estado *detached HEAD* ❶. *HEAD* es el estado comprometido actual del proyecto; estás en *detached* porque has abandonado una rama con nombre (`main` , en este caso).

When you check out a previous commit, you leave the main branch and enter what Git refers to as a *detached HEAD* state ❶. *HEAD* is the current committed state of the project; you're *detached* because you've left a named branch (`main` , in this case).

Para volver a la rama `main` , sigue la sugerencia ❷ para deshacer la operación anterior:

To get back to the `main` branch, you follow the suggestion ❷ to undo the previous operation:

```
git_practice$ git switch -
Previous HEAD position was cea13d Started project.
Switched to branch 'main'
git_practice$
```

Este comando te devuelve a la rama `main` . A menos que quieras trabajar con algunas funciones más avanzadas de Git, lo mejor es que no hagas ningún cambio en tu proyecto cuando hayas comprobado una confirmación anterior. Sin embargo, si eres el único que trabaja en un proyecto y quieres descartar todas las confirmaciones más recientes y volver a un estado anterior, puedes restablecer el proyecto a una confirmación anterior. Trabajando desde la rama `main` , introduce lo siguiente:

This command brings you back to the `main` branch. Unless you want to work with some more advanced features of Git, it's best not to make any changes to your project when you've checked out a previous commit. However, if you're the only one working on a project and you want to discard all of the more recent commits and go back to a previous state, you can reset the project to a previous commit. Working from the `main` branch, enter the following:

```
❶ git_practice$ git status
On branch main
nothing to commit, working directory clean
❷ git_practice$ git log --pretty=oneline
945fa13af128a266d0114eebb7a3276f7d58ecd2 (HEAD -> main) Extended greeting.
cea13ddc51b885d05a410201a54faf20e0d2e246 Started project.
❸ git_practice$ git reset --hard cea13d
HEAD is now at cea13dd Started project.
❹ git_practice$ git status
On branch main
nothing to commit, working directory clean
❺ git_practice$ git log --pretty=oneline
cea13ddc51b885d05a410201a54faf20e0d2e246 (HEAD -> main) Started project.
git_practice$
```

Primero comprobamos el estado para asegurarnos de que estamos en la rama `main` ❶ . Cuando miramos el log, vemos ambos commits ❷ . A continuación, emitimos el comando `git reset --hard` con los seis primeros caracteres del ID de referencia de el commit al que queremos volver permanentemente ❸ . Comprobamos de nuevo el estado y vemos que estamos en la rama `main` sin nada que confirmar ❹ . Cuando volvemos a mirar el registro, vemos que estamos en el commit desde el que queríamos volver a empezar ❺ .

We first check the status to make sure we're on the `main` branch ❶ . When we look at the log, we see both commits ❷ . We then issue the `git reset --hard` command with the first six characters of the reference ID of the commit we want to go back to permanently ❸ . We check the status again and see we're on the `main` branch with nothing to commit ❹ .

When we look at the log again, we see that we're at the commit we wanted to start over from ❸ .

Borrar el repositorio (*Deleting the Repository*)

A veces estropearás el historial de tu repositorio y no sabrás cómo recuperarlo. Si esto ocurre, primero considera la posibilidad de pedir ayuda utilizando los métodos comentados en el Apéndice C . Si no puedes arreglarlo y estás trabajando en un proyecto en solitario, puedes seguir trabajando con los archivos pero deshacerte del historial del proyecto borrando el directorio `.git` . Esto no afectará al estado actual de ninguno de los archivos, pero borrará todos los commits, por lo que no podrás consultar ningún otro estado del proyecto.

Sometimes you'll mess up your repository's history and won't know how to recover it. If this happens, first consider asking for help using the approaches discussed in Appendix C . If you can't fix it and you're working on a solo project, you can continue working with the files but get rid of the project's history by deleting the `.git` directory. This won't affect the current state of any of the files, but it will delete all commits, so you won't be able to check out any other states of the project.

Para ello, abre un explorador de archivos y borra el repositorio `.git` o bórralo desde la línea de comandos. Después, tendrás que empezar de nuevo con un repositorio nuevo para empezar a seguir tus cambios de nuevo. Este es el aspecto de todo este proceso en una sesión de terminal:

To do this, either open a file browser and delete the `.git` repository or delete it from the command line. Afterward, you'll need to start over with a fresh repository to start tracking your changes again. Here's what this entire process looks like in a terminal session:

```
❶ git_practice$ git status
On branch main
nothing to commit, working directory clean
❷ git_practice$ rm -rf .git/
❸ git_practice$ git status
fatal: Not a git repository (or any of the parent directories): .git
❹ git_practice$ git init
Initialized empty Git repository in git_practice/.git/
❺ git_practice$ git status
```

On branch main
No commits yet

Untracked files:

(use "git add <file>..." to include in what will be committed)

.gitignore
hello_git.py

nothing added to commit but untracked files present (use "git add" to track)

⑥ git_practice\$ git add .

git_practice\$ git commit -m "Starting over."

[main (root-commit) 14ed9db] Starting over.

2 files changed, 5 insertions(+)

create mode 100644 .gitignore

create mode 100644 hello_git.py

⑦ git_practice\$ git status

On branch main

nothing to commit, working tree clean

git_practice\$

Primero comprobamos el estado y vemos que tenemos un directorio de trabajo limpio ① . Después utilizamos el comando `rm -rf .git/` para borrar el directorio `.git` (del `.git` en Windows) ② . Cuando comprobamos el estado después de borrar la carpeta `.git` , se nos indica que no se trata de un repositorio Git ③ . Toda la información que Git utiliza para rastrear un repositorio se almacena en la carpeta `.git` , por lo que al eliminarla se elimina todo el repositorio.

We first check the status and see that we have a clean working directory ① . Then we use the command `rm -rf .git/` to delete the `.git` directory (del `.git` on Windows) ② . When we check the status after deleting the `.git` folder, we're told that this is not a Git repository ③ . All the information Git uses to track a repository is stored in the `.git` folder, so removing it deletes the entire repository.

Entonces podemos utilizar `git init` para iniciar un nuevo repositorio ④ . Comprobando el estado vemos que estamos de nuevo en la fase inicial, esperando la primera confirmación ⑤ . Añadimos los archivos y hacemos la primera confirmación ⑥ . Comprobar el estado ahora nos muestra que estamos en la nueva rama `main` sin nada que confirmar ⑦ .

We're then free to use `git init` to start a fresh repository ④ . Checking the status shows that we're back at the initial stage, awaiting the first commit ⑤ . We add the files and

make the first commit ⑥ . Checking the status now shows us that we're on the new main branch with nothing to commit ⑦ .

Utilizar el control de versiones requiere un poco de práctica, pero una vez que empieces a usarlo, nunca querrás volver a trabajar sin él.

Using version control takes a bit of practice, but once you start using it, you'll never want to work without it again.

Apéndice E: Solución de problemas de implementación (Appendix E: Troubleshooting Deployments)

E

Solución de problemas de despliegue *(E Troubleshooting Deployments)*



Desplegar una aplicación es tremendamente satisfactorio cuando funciona, sobre todo si nunca lo has hecho antes. Sin embargo, pueden surgir muchos obstáculos en el proceso de despliegue y, por desgracia, algunos de estos problemas pueden ser difíciles de identificar y solucionar. Este apéndice te ayudará a comprender los enfoques modernos de la implantación y te dará formas concretas de solucionar los problemas del proceso de implantación cuando las cosas no funcionen.

Deploying an app is tremendously satisfying when it works, especially if you've never done it before. However, there are many obstacles that can arise in the deployment process, and unfortunately, some of these issues can be difficult to identify and address. This appendix

will help you understand modern approaches to deployment and give you specific ways to troubleshoot the deployment process when things aren't working.

Si la información adicional de este apéndice no es suficiente para ayudarte a llevar a cabo con éxito el proceso de despliegue, consulta los recursos en línea en https://ehmatthes.github.io/pcc_3e ; las actualizaciones que hay allí te ayudarán casi con toda seguridad a llevar a cabo un despliegue con éxito.

If the additional information in this appendix isn't enough to help you get through the deployment process successfully, see the online resources at https://ehmatthes.github.io/pcc_3e ; the updates there will almost certainly help you carry out a successful deployment.

Comprender las implantaciones (*Understanding Deployments*)

Cuando trates de solucionar un intento de despliegue concreto, es útil tener una idea clara de cómo funciona un despliegue típico. *Deployment* se refiere al proceso de tomar un proyecto que funciona en tu sistema local, y copiar ese proyecto a un servidor remoto de forma que pueda responder a las peticiones de cualquier usuario de Internet. El entorno remoto difiere de un sistema local típico en varios aspectos importantes: probablemente no sea el mismo sistema operativo (SO) que el que estás utilizando, y lo más probable es que sea uno de los muchos servidores virtuales de un único servidor físico.

When you're trying to troubleshoot a particular deployment attempt, it's helpful to have a clear understanding of how a typical deployment works. *Deployment* refers to the process of taking a project that works on your local system, and copying that project to a remote server in a way that allows it to respond to requests from any user on the internet. The remote environment differs from a typical local system in a number of important ways: it's probably not the same operating system (OS) as the one you're using, and it's most likely one of many virtual servers on a single physical server.

Cuando despliegues un proyecto, o *push* al servidor remoto, deberás seguir los siguientes pasos:

When you deploy a project, or *push* it to the remote server, the following steps need to be taken:

Listado en español:

- Crear un servidor virtual en una máquina física en un centro de datos.
- Establecer una conexión entre el sistema local y el servidor remoto.
- Copia el código del proyecto en el servidor remoto.
- Identifica todas las dependencias del proyecto e instálalas en el servidor remoto.
- Configura una base de datos y ejecuta las migraciones existentes.
- Copia los archivos estáticos (CSS, archivos JavaScript y archivos multimedia) en un lugar donde se puedan servir de forma eficiente.
- Inicia un servidor para gestionar las solicitudes entrantes.
- Comienza a dirigir las solicitudes entrantes al proyecto, una vez que esté preparado para gestionarlas.

Listing in English:

- Create a virtual server on a physical machine at a datacenter.
- Establish a connection between the local system and the remote server.
- Copy the project's code to the remote server.
- Identify all of the project's dependencies and install them on the remote server.
- Set up a database and run any existing migrations.
- Copy static files (CSS, JavaScript files, and media files) to a place where they can be served efficiently.
- Start a server to handle incoming requests.
- Start routing incoming requests to the project, once it's ready to handle requests.

Si tienes en cuenta todo lo que conlleva un despliegue, no es de extrañar que los despliegues fracasen a menudo. Afortunadamente, una vez que comprendas lo que debería estar ocurriendo, tendrás más posibilidades de identificar lo que ha ido mal. Si puedes identificar lo que ha ido mal, tal vez puedas identificar una solución que haga que el siguiente intento de despliegue tenga éxito.

When you consider all that goes into a deployment, it's no wonder deployments often fail. Fortunately, once you gain an understanding of what should be happening, you'll stand a better chance of identifying what went wrong. If you can identify what went wrong, you might be able to identify a fix that will make the next deployment attempt successful.

Puedes desarrollar localmente en un tipo de SO y enviar a un servidor que ejecute un SO diferente. Es importante que sepas a qué tipo de sistema estás enviando el software, porque eso puede ayudarte a solucionar algunos problemas. En el momento de escribir esto, un servidor remoto básico en Platform.sh ejecuta Debian Linux; la mayoría de los servidores remotos son sistemas basados en Linux.

You can develop locally on one kind of OS and push to a server running a different OS. It's important to know what kind of system you're pushing to, because that can inform some of your troubleshooting work. At the time of this writing, a basic remote server on Platform.sh runs Debian Linux; most remote servers are Linux-based systems.

Solución de problemas básicos (*Basic Troubleshooting*)

Algunos pasos de la resolución de problemas son específicos de cada SO, pero llegaremos a eso dentro de un momento. En primer lugar, vamos a considerar los pasos que todo el mundo debería probar al solucionar los problemas de una implantación.

Some troubleshooting steps are specific to each OS, but we'll get to that in a moment. First, let's consider the steps everyone should try when troubleshooting a deployment.

Tu mejor recurso es la salida generada durante el intento de inserción. Esta salida puede parecer intimidante; si eres nuevo en el despliegue de aplicaciones, puede parecer muy técnica, y suele haber mucha. La buena noticia es que no necesitas entenderlo todo. Deberías tener dos objetivos al hojear la salida del registro: identificar los pasos de despliegue que funcionaron e identificar los pasos que no funcionaron. Si lo consigues, en podrás averiguar qué cambiar en tu proyecto, o en tu proceso de despliegue, para que tu próximo empuje sea un éxito.

Your best resource is the output generated during the attempted push. This output can look intimidating; if you're new to deploying apps, it can look highly technical, and there's usually a lot of it. The good news is you don't need to understand everything in the output. You should have two goals when skimming log output: identify any deployment steps that worked, and identify any steps that didn't. If you can do this, you might be able to figure out what to change in your project, or in your deployment process, to make your next push successful.

Sigue las sugerencias en pantalla (*Follow Onscreen Suggestions*)

A veces, la plataforma a la que estás empujando generará un mensaje con una sugerencia clara sobre cómo solucionar el problema. Por ejemplo, éste es el mensaje que verás si creas un proyecto Platform.sh antes de inicializar un repositorio Git, y luego intentas empujar el proyecto:

Sometimes, the platform you're pushing to will generate a message that has a clear suggestion for how to address the issue. For example, here's the message you'll see if you create a Platform.sh project before initializing a Git repository, and then try to push the project:

```
$ platform push
```

```
❶ Enter a number to choose a project:
```

```
[0] ll_project (votohz445ljyg)
```

```
> 0
```

```
❷ [RootNotFoundException]
```

```
Project root not found. This can only be run from inside a project directory.
```

```
❸ To set the project for this Git repository, run:
```

```
platform project:set-remote [id]
```

Estamos intentando empujar un proyecto, pero el proyecto local aún no se ha asociado a un proyecto remoto. Así que la CLI de Platform.sh nos pregunta qué proyecto remoto queremos empujar a ❶ . Introducimos 0 , para seleccionar el único proyecto de la lista. Pero a continuación, vemos un `RootNotFoundException` ❷ . Esto ocurre porque Platform.sh busca un directorio `.git` cuando inspecciona el proyecto local, para averiguar cómo conectar el proyecto local con el proyecto remoto. En este caso, como no había ningún directorio `.git` cuando se creó el proyecto remoto, esa conexión nunca se estableció. La CLI sugiere una solución ❸ ; nos está diciendo que podemos especificar el proyecto remoto que debe asociarse a este proyecto local, utilizando el comando `project:set-remote` .

We're trying to push a project, but the local project hasn't been associated with a remote project yet. So, the Platform.sh CLI asks which remote project we want to push to ❶ . We enter 0 , to select the only project listed. But next, we see a `RootNotFoundException` ❷ . This happens because Platform.sh looks for a `.git` directory when it inspects the local project, to figure out how to connect the local project with the remote project. In this case, since there was no `.git` directory when the remote project was created, that connection was never established. The CLI suggests a fix ❸ ; it's telling us that we can specify the remote project that should be associated with this local project, using the `project:set-remote` command.

Probemos esta sugerencia:

Let's try this suggestion:

```
$ platform project:set-remote votohz445ljyg
Setting the remote project for this repository to: ll_project (votohz445ljyg)
```

The remote project for this repository is
now set to: ll_project (votohz445ljyg)

En la salida anterior, la CLI mostraba el ID de este proyecto remoto, votohz4451jyg . Así que ejecutamos el comando sugerido, utilizando este ID, y la CLI es capaz de establecer la conexión entre el proyecto local y el proyecto remoto.

In the previous output, the CLI showed the ID of this remote project, votohz4451jyg . So we run the command that's suggested, using this ID, and the CLI is able to make the connection between the local project and the remote project.

Ahora vamos a intentar empujar el proyecto de nuevo:

Now let's try to push the project again:

```
$ platform push
Are you sure you want to push to the main (production) branch? [Y/n] y
Pushing HEAD to the existing environment main
--snip--
```

Esto ha sido un push con éxito; seguir la sugerencia en pantalla ha funcionado.

This was a successful push; following the onscreen suggestion worked.

Debes tener cuidado al ejecutar comandos que no entiendas completamente. Sin embargo, si tienes buenas razones para creer que un comando puede hacer poco daño, y si confías en la fuente de la recomendación, puede ser razonable probar las sugerencias que ofrecen las

herramientas que estás utilizando.

You should be careful about running commands that you don't fully understand. However, if you have good reason to believe that a command can do little harm, and if you trust the source of the recommendation, it might be reasonable to try the suggestions offered by the tools you're using.

Nota (Note)

Ten en cuenta que hay personas que te dirán que ejecutes comandos que borrarán tu sistema o lo expondrán a la explotación remota. Seguir las sugerencias de una herramienta proporcionada por una empresa u organización en la que confías es diferente de seguir las sugerencias de personas al azar en Internet. Siempre que trates con conexiones remotas, procede con mucha precaución.

Keep in mind there are individuals who will tell you to run commands that will wipe your system or expose your system to remote exploitation. Following the suggestions of a tool provided by a company or organization you trust is different from following the suggestions of random people online. Anytime you're dealing with remote connections, proceed with an abundance of caution.

Lee el registro de salida (Read the Log Output)

Como ya se ha dicho, la salida del registro que ves cuando ejecutas un comando como `platform push` puede ser a la vez informativa e intimidatoria. Lee el siguiente fragmento de la salida del registro, tomado de un intento diferente de utilizar `platform push`, y comprueba si puedes detectar el problema:

As mentioned earlier, the log output that you see when you run a command like `platform push` can be both informative and intimidating. Read through the following snippet of log output, taken from a different attempt at using `platform push`, and see if you can spot the issue:

```
--snip--
Collecting soupsieve==2.3.2.post1
  Using cached soupsieve-2.3.2.post1-py3-none-any.whl (37 kB)
Collecting sqlparse==0.4.2
  Using cached sqlparse-0.4.2-py3-none-any.whl (42 kB)
Installing collected packages: platformshconfig, sqlparse,...
Successfully installed Django-4.1 asgiref-3.5.2 beautifulsoup4-4.11.1...
W: ERROR: Could not find a version that satisfies the requirement gunicorn
W: ERROR: No matching distribution found for gunicorn
```

130 static files copied to '/app/static'.

Executing pre-flight checks...

```
--snip--
```

Cuando falla un intento de despliegue, una buena estrategia consiste en examinar la salida del registro y ver si puedes detectar algo que parezca una advertencia o un error. Las advertencias son bastante comunes; a menudo son mensajes sobre los próximos cambios en las dependencias de un proyecto, para ayudar a los desarrolladores a abordar los problemas antes de que causen fallos reales.

When a deployment attempt fails, a good strategy is to look through the log output and see if you can spot anything that looks like warnings or errors. Warnings are fairly common; they're often messages about upcoming changes in a project's dependencies, to help developers address issues before they cause actual failures.

Un push correcto puede tener advertencias, pero no debería tener errores. En este caso, Platform.sh no pudo encontrar la forma de instalar el requisito `gunicornn`. Se trata de un error tipográfico en el archivo `requirements_remote.txt`, que debía incluir `gunicorn` (con un `r`). No siempre es fácil detectar el problema de raíz en la salida del registro, especialmente cuando el problema causa un montón de errores y advertencias en cascada. Al igual que cuando lees un rastreo en tu sistema local, es una buena idea fijarse bien en los primeros errores que aparecen en la lista, y también en los últimos. La mayoría de los errores intermedios suelen ser paquetes internos que se quejan de que algo ha ido mal, y pasan mensajes sobre el error a otros paquetes internos. El error real que podemos solucionar suele ser uno de los primeros o últimos errores de la lista.

A successful push may have warnings, but it shouldn't have any errors. In this case, Platform.sh couldn't find a way to install the requirement `gunicornn`. This is a typo in the `requirements_remote.txt` file, which was supposed to include `gunicorn` (with one `r`). It's

not always easy to spot the root issue in log output, especially when the problem causes a bunch of cascading errors and warnings. Just like when reading a traceback on your local system, it's a good idea to look closely at the first few errors that are listed, and also the last few errors. Most of the errors in between tend to be internal packages complaining that something went wrong, and passing messages about the error to other internal packages. The actual error we can fix is usually one of the first or last errors listed.

A veces, serás capaz de detectar el error, y otras veces, no tendrás ni idea de lo que significa la salida. Sin duda merece la pena intentarlo, y utilizar la salida del registro para diagnosticar con éxito un error es una sensación tremendamente satisfactoria. A medida que pases más tiempo examinando los resultados del registro, mejorarás en la identificación de la información más significativa para ti.

Sometimes, you'll be able to spot the error, and other times, you'll have no idea what the output means. It's certainly worth a try, and using log output to successfully diagnose an error is a tremendously satisfying feeling. As you spend more time looking through log output, you'll get better at identifying the information that's most meaningful to you.

Solución de problemas específicos del sistema operativo (OS-Specific Troubleshooting)

Puedes desarrollar en el sistema operativo que quieras y enviar a cualquier host que desees. Las herramientas para enviar proyectos se han desarrollado lo suficiente como para modificar tu proyecto según sea necesario para que se ejecute correctamente en el sistema remoto. Sin embargo, pueden surgir algunos problemas específicos del sistema operativo.

You can develop on any operating system you like and push to any host you like. The tools for pushing projects have developed enough that they'll modify your project as needed to run correctly on the remote system. However, there are some OS-specific issues that can arise.

En el proceso de despliegue de Platform.sh, una de las fuentes más probables de dificultades es la instalación de la CLI. Aquí tienes el comando para hacerlo:

In the Platform.sh deployment process, one of the most likely sources of difficulties is installing the CLI. Here's the command to do so:

```
$ curl -fsS https://platform.sh/cli/installer | php
```

El comando comienza con `curl`, una herramienta que te permite solicitar recursos remotos, a los que se accede a través de una URL, dentro de un terminal. En este caso, se utiliza para descargar el instalador de la CLI desde un servidor Platform.sh. La sección `-fsS` del comando es un conjunto de banderas que modifican cómo se ejecuta `curl`. La bandera `f` indica a `curl` que suprima la mayoría de los mensajes de error, para que el instalador de la CLI pueda gestionarlos en lugar de informarte de todos ellos. La bandera `s` indica a `curl` que se ejecute en silencio; deja que el instalador de la CLI decida qué información mostrar en el terminal. La bandera `S` indica a `curl` que muestre un mensaje de error si el comando general falla. El `| php` al final del comando indica a tu sistema que ejecute el archivo de instalación descargado utilizando un intérprete de PHP, porque la CLI Platform.sh está escrita en PHP.

The command starts with `curl`, a tool that lets you request remote resources, accessed through a URL, within a terminal. Here, it's being used to download the CLI installer from a Platform.sh server. The `-fsS` section of the command is a set of flags that modify how `curl` runs. The `f` flag tells `curl` to suppress most error messages, so the CLI installer can handle them instead of reporting them all to you. The `s` flag tells `curl` to run silently; it lets the CLI installer decide what information to show in the terminal. The `S` flag tells `curl` to show an error message if the overall command fails. The `| php` at the end of the command tells your system to run the downloaded installer file using a PHP interpreter, because the Platform.sh CLI is written in PHP.

Esto significa que tu sistema necesita `curl` y PHP para instalar la CLI Platform.sh. Para utilizar la CLI, también necesitarás Git y un terminal que pueda ejecutar comandos Bash. *Bash* es un lenguaje que está disponible en la mayoría de los entornos de servidor. La mayoría de los sistemas modernos tienen espacio suficiente para instalar varias herramientas como ésta.

This means your system needs `curl` and PHP in order to install the Platform.sh CLI. To use the CLI, you'll also need Git, and a terminal that can run Bash commands. *Bash* is a language that's available in most server environments. Most modern systems have plenty of room for multiple tools like this to be installed.

Las siguientes secciones te ayudarán a abordar estos requisitos para tu sistema operativo. Si aún no tienes instalado Git, consulta las instrucciones para instalar Git en la página 484

del Apéndice D y luego ve a la sección de aquí que sea aplicable a tu SO.

The following sections will help you address these requirements for your OS. If you don't already have Git installed, see the instructions for installing Git on page 484 in Appendix D and then go to the section here that's applicable to your OS.

Nota (Note)

Una herramienta excelente para entender comandos de terminal como el que se muestra aquí es <https://explainshell.com>. Introduce el comando que intentas comprender y el sitio te mostrará la documentación de todas las partes del comando. Pruébalo con el comando utilizado para instalar la CLI Platform.sh.

An excellent tool for understanding terminal commands like the one shown here is <https://explainshell.com>. Enter the command you're trying to understand, and the site will show you the documentation for all the parts of your command. Try it out with the command used to install the Platform.sh CLI.

Despliegue desde Windows (*Deploying from Windows*)

Windows ha experimentado un resurgimiento de su popularidad entre los programadores en los últimos años. Windows ha integrado muchos elementos diferentes de otros sistemas operativos, proporcionando a los usuarios una serie de opciones sobre cómo realizar el trabajo de desarrollo local e interactuar con sistemas remotos.

Windows has seen a resurgence in popularity with programmers in recent years. Windows has integrated many different elements of other operating systems, providing users with a number of options for how to do local development work and interact with remote systems.

Una de las dificultades más significativas de la implantación desde Windows es que el sistema operativo base de Windows no es el mismo que utiliza un servidor remoto basado en Linux. Un sistema Windows base tiene un conjunto de herramientas y lenguajes diferente al de un sistema Linux base, por lo que, para llevar a cabo el trabajo de despliegue

desde Windows, tendrás que elegir cómo integrar los conjuntos de herramientas basados en Linux en tu entorno local.

One of the most significant difficulties in deploying from Windows is that the core Windows operating system is not the same as what a Linux-based remote server uses. A base Windows system has a different set of tools and languages than a base Linux system, so to carry out deployment work from Windows, you'll need to choose how to integrate Linux-based tool sets into your local environment.

Subsistema Windows para Linux (Windows Subsystem for Linux)

Un enfoque popular es utilizar *Windows Subsystem for Linux* (*WSL*), un entorno que permite ejecutar Linux directamente en Windows. Si tienes configurado WSL, utilizar la CLI Platform.sh en Windows será tan fácil como utilizarla en Linux. La CLI no sabrá que se está ejecutando en Windows; sólo verá el entorno Linux en el que la estás utilizando.

One popular approach is to use *Windows Subsystem for Linux* (*WSL*), an environment that allows Linux to run directly on Windows. If you have WSL set up, using the Platform.sh CLI on Windows becomes as easy as using it on Linux. The CLI won't know it's running on Windows; it will just see the Linux environment you're using it in.

Configurar WSL es un proceso de dos pasos: primero instalas WSL y luego eliges una distribución de Linux para instalarla en el entorno WSL. Configurar un entorno WSL es más de lo que se puede describir aquí; si te interesa este enfoque y aún no lo tienes configurado, consulta la documentación en <https://docs.microsoft.com/en-us/windows/wsl/about> . Una vez que tengas configurado WSL, puedes seguir las instrucciones de la sección Linux de este apéndice para continuar tu trabajo de despliegue.

Setting up WSL is a two-step process: you first install WSL, and then choose a Linux distribution to install into the WSL environment. Setting up a WSL environment is more than can be described here; if you're interested in this approach and don't already have it set up, see the documentation at <https://docs.microsoft.com/en-us/windows/wsl/about> . Once you have WSL set up, you can follow the instructions in the Linux section of this appendix to continue your deployment work.

Git Bash (Git Bash)

Otro enfoque para crear un entorno local desde el que puedas desplegar utiliza *Git Bash* , un entorno de terminal compatible con Bash pero que se ejecuta en Windows. Git Bash se instala junto con Git cuando utilizas el instalador de <https://git-scm.com> . Este enfoque puede funcionar, pero no es tan ágil como WSL. En este enfoque, tendrás que utilizar un terminal Windows para algunos pasos y un terminal Git Bash para otros.

Another approach to building a local environment that you can deploy from uses *Git Bash* , a terminal environment that's compatible with Bash but runs on Windows. Git Bash is installed along with Git when you use the installer from <https://git-scm.com> . This approach can work, but it isn't as streamlined as WSL. In this approach, you'll have to use a Windows terminal for some steps and a Git Bash terminal for others.

Primero tendrás que instalar PHP. Puedes hacerlo con *XAMPP* , un paquete que incluye PHP con algunas otras herramientas para desarrolladores. Ve a <https://apachefriends.org> y haz clic en el botón para descargar XAMPP para Windows. Abre el instalador y ejecútalo; si ves una advertencia sobre las restricciones del Control de Cuentas de Usuario (UAC), haz clic en **Aceptar** . Acepta todos los valores predeterminados del instalador.

First you'll need to install PHP. You can do this with *XAMPP* , a package that bundles PHP with a few other developer-focused tools. Go to <https://apachefriends.org> and click the button to download XAMPP for Windows. Open the installer and run it; if you see a warning about User Account Control (UAC) restrictions, click **OK** . Accept all of the installer's defaults.

Cuando termine de ejecutarse el instalador, tendrás que añadir PHP a la ruta de tu sistema; esto indicará a Windows dónde buscar cuando quieras ejecutar PHP. En el menú Inicio, escribe `path` y haz clic en **Editar las variables de entorno del sistema** ; haz clic en el botón **Variables de entorno** . Deberías ver la variable `Path` resaltada; haz clic en **Editar** en este panel. Haz clic en **Nueva** para añadir una nueva ruta a la lista actual de rutas. Suponiendo que mantuviste la configuración predeterminada al ejecutar el instalador de XAMPP, añade `C:\xampp\php` en el cuadro que aparece y haz clic en **Aceptar** . Cuando hayas terminado, cierra todos los cuadros de diálogo del sistema que sigan abiertos.

When the installer finishes running, you'll need to add PHP to your system's path; this will tell Windows where to look when you want to run PHP. In the Start menu, enter `path` and click **Edit the System Environment Variables** ; click the button labeled **Environment**

Variables . You should see the variable `Path` highlighted; click **Edit** under this pane. Click **New** to add a new path to the current list of paths. Assuming you kept the default settings when running the XAMPP installer, add `C:\xampp\php` in the box that appears, then click **OK** . When you're finished, close all of the system dialogs that are still open.

Una vez cumplidos estos requisitos, puedes instalar la CLI Platform.sh. Necesitarás utilizar un terminal de Windows con privilegios de administrador; introduce `command` en el menú Inicio, y en la aplicación Símbolo del sistema, haz clic en **Ejecutar como administrador** . En el terminal que aparece, introduce el siguiente comando:

With these requirements taken care of, you can install the Platform.sh CLI. You'll need to use a Windows terminal with administrator privileges; enter `command` into the Start menu, and under the Command Prompt app, click **Run as administrator** . In the terminal that appears, enter the following command:

```
> curl -fsS https://platform.sh/cli/installer | php
```

Esto instalará la CLI Platform.sh, como se ha descrito anteriormente.

This will install the Platform.sh CLI, as described earlier.

Por último, trabajarás en Git Bash. Para abrir un terminal Git Bash, ve al menú Inicio y busca `git bash` . Haz clic en la **aplicación Git Bash** que aparece; deberías ver que se abre una ventana de terminal. Puedes utilizar comandos tradicionales basados en Linux como `ls` en este terminal, así como comandos basados en Windows como `dir` . Para asegurarte de que la instalación se ha realizado correctamente, ejecuta el comando `platform list` . Deberías ver una lista de todos los comandos de la CLI Platform.sh. A partir de este momento, realiza todo tu trabajo de despliegue utilizando la CLI Platform.sh dentro de una ventana de terminal Git Bash.

Finally, you'll work in Git Bash. To open a Git Bash terminal, go to the Start menu and search for `git bash` . Click the **Git Bash app** that appears; you should see a terminal window open. You can use traditional Linux-based commands like `ls` in this terminal, as well as Windows-based commands like `dir` . To make sure the installation was successful, issue the `platform list` command. You should see a list of all the commands in the

Platform.sh CLI. From this point forward, carry out all of your deployment work using the Platform.sh CLI inside a Git Bash terminal window.

Despliegue desde macOS (*Deploying from macOS*)

El sistema operativo macOS no está basado en Linux, pero ambos se desarrollaron sobre principios similares. Lo que esto significa, en la práctica, es que muchos de los comandos y flujos de trabajo que utilizas en macOS funcionarán también en un entorno de servidor remoto. Puede que necesites instalar algunos recursos orientados al desarrollador para disponer de todas estas herramientas en tu entorno local de macOS. Si en algún momento de tu trabajo se te pide que instales la *command line developer tools*, haz clic en **Instalar** para aprobar la instalación.

The macOS operating system is not based on Linux, but they were both developed on similar principles. What this means, practically, is that a lot of the commands and workflows that you use on macOS will work in a remote server environment as well. You might need to install some developer-focused resources in order to have all of these tools available in your local macOS environment. If you get a prompt to install the *command line developer tools* at any point in your work, click **Install** to approve the installation.

La dificultad más probable al instalar la CLI Platform.sh es asegurarse de que PHP está instalado. Si aparece un mensaje que indica que no se encuentra el comando `php`, tendrás que instalar PHP. Una de las formas más sencillas de instalar PHP es utilizando el gestor de paquetes *Homebrew*, que facilita la instalación de una amplia variedad de paquetes de los que dependen los programadores. Si aún no tienes Homebrew instalado, visita <https://brew.sh> y sigue las instrucciones para instalarlo.

The most likely difficulty when installing the Platform.sh CLI is making sure PHP is installed. If you see a message that the `php` command is not found, you'll need to install PHP. One of the easiest ways to install PHP is by using the *Homebrew* package manager, which facilitates the installation of a wide variety of packages that programmers depend on. If you don't already have Homebrew installed, visit <https://brew.sh> and follow the instructions to install it.

Una vez instalado Homebrew, utiliza el siguiente comando para instalar PHP:

Once Homebrew is installed, use the following command to install PHP:

```
$ brew install php
```

Esto tardará un rato en ejecutarse, pero una vez que se haya completado, deberías poder instalar correctamente la CLI Platform.sh.

This will take a while to run, but once it has completed, you should be able to successfully install the Platform.sh CLI.

Despliegue desde Linux *(Deploying from Linux)*

Dado que la mayoría de los entornos de servidor están basados en Linux, no deberías tener muchas dificultades para instalar y utilizar la CLI Platform.sh. Si intentas instalar la CLI en un sistema con una instalación reciente de Ubuntu, te dirá exactamente qué paquetes necesitas:

Because most server environments are Linux-based, you should have very little difficulty installing and using the Platform.sh CLI. If you try to install the CLI on a system with a fresh installation of Ubuntu, it will tell you exactly which packages you need:

```
$ curl -fsS https://platform.sh/cli/installer | php
Command 'curl' not found, but can be installed with:
sudo apt install curl
Command 'php' not found, but can be installed with:
sudo apt install php-cli
```

La salida real tendrá más información sobre algunos otros paquetes que podrían funcionar, además de alguna información sobre la versión. El siguiente comando instalará curl y PHP:

The actual output will have more information about a few other packages that would work, plus some version information. The following command will install curl and PHP:

```
$ sudo apt install curl php-cli
```

Tras ejecutar este comando, el comando de instalación CLI Platform.sh debería ejecutarse correctamente. Dado que tu entorno local es bastante similar a la mayoría de los entornos

de alojamiento basados en Linux, gran parte de lo que aprendas sobre cómo trabajar en tu terminal se trasladará también al trabajo en un entorno remoto.

After running this command, the Platform.sh CLI installation command should run successfully. Since your local environment is quite similar to most Linux-based hosting environments, much of what you learn about working in your terminal will carry over to working in a remote environment as well.

Otros métodos de instalación *(Other Deployment Approaches)*

Si Platform.sh no te funciona, o si quieres probar un enfoque diferente, hay muchas plataformas de alojamiento entre las que elegir. Algunas funcionan de forma similar al proceso descrito en el Capítulo 20 , y otras tienen un enfoque muy diferente para llevar a cabo los pasos descritos al principio de este apéndice:

If Platform.sh doesn't work for you, or if you want to try a different approach, there are many hosting platforms to choose from. Some work similarly to the process described in Chapter 20 , and some have a much different approach to carrying out the steps described at the beginning of this appendix:

Listado en español:

- Platform.sh te permite utilizar un navegador para llevar a cabo los pasos para los que utilizamos la CLI. Si te gustan más las interfaces basadas en navegador que los flujos de trabajo basados en terminal, puede que prefieras este enfoque.
- Hay otros proveedores de alojamiento que ofrecen enfoques basados tanto en la CLI como en el navegador. Algunos de estos proveedores ofrecen terminales dentro de su navegador, por lo que no tienes que instalar nada en tu sistema.
- Algunos proveedores te permiten enviar tu proyecto a un sitio de alojamiento de código remoto como GitHub, y luego conectar tu repositorio de GitHub al sitio de alojamiento. El servidor extrae entonces tu código de GitHub, en lugar de pedirte que envíes tu código desde tu sistema local directamente al servidor. Platform.sh también admite este tipo de flujo de trabajo.
- Algunos proveedores ofrecen una serie de servicios entre los que puedes elegir para crear una infraestructura que se adapte a tu proyecto. Esto normalmente requiere que tengas un conocimiento más profundo del proceso de despliegue, y de lo que necesita un servidor remoto para servir a un proyecto. Estos hosts incluyen Amazon Web Services (AWS) y la plataforma Azure de Microsoft. Puede ser mucho más difícil hacer un seguimiento de tus costes en este tipo de plataformas, porque cada servicio puede acumular cargos de forma independiente.

- Mucha gente aloja sus proyectos en un servidor virtual privado (VPS). En este enfoque, alquilas un servidor virtual que actúa como un ordenador remoto, te conectas al servidor, instalas el software necesario para ejecutar tu proyecto, copias tu código, estableces las conexiones adecuadas y permites que tu servidor empiece a aceptar peticiones.

Listing in English:

- Platform.sh allows you to use a browser to carry out the steps we used the CLI for. If you like browser-based interfaces better than terminal-based workflows, you may prefer this approach.
- There are a number of other hosting providers that offer both CLI- and browser-based approaches. Some of these providers offer terminals within their browser, so you don't have to install anything on your system.
- Some providers allow you to push your project to a remote code hosting site like GitHub, and then connect your GitHub repository to the hosting site. The host then pulls your code from GitHub, instead of requiring you to push your code from your local system directly to the host. Platform.sh supports this kind of workflow as well.
- Some providers offer an array of services that you select from, in order to put together an infrastructure that works for your project. This typically requires you to have a deeper understanding of the deployment process, and what a remote server needs in order to serve a project. These hosts include Amazon Web Services (AWS) and Microsoft's Azure platform. It can be much harder to track your costs in these kinds of platforms, because each service can accrue charges independently.
- Many people host their projects on a virtual private server (VPS). In this approach, you rent a virtual server that acts just like a remote computer, log in to the server, install the software needed to run your project, copy your code over, set the right connections, and allow your server to start accepting requests.

Regularmente aparecen nuevas plataformas y enfoques de alojamiento; encuentra una que te parezca atractiva, e invierte tiempo en aprender el proceso de despliegue de ese proveedor. Mantén tu proyecto el tiempo suficiente para que llegues a saber qué funciona bien con el enfoque de tu proveedor y qué no. Ninguna plataforma de alojamiento va a ser perfecta; tendrás que juzgar continuamente si el proveedor que utilizas actualmente es lo suficientemente bueno para tu caso de uso.

New hosting platforms and approaches appear on a regular basis; find one that looks appealing to you, and invest the time to learn that provider's deployment process. Maintain your project long enough so that you get to know what works well with your provider's approach and what doesn't. No hosting platform is going to be perfect; you'll need to make an ongoing judgement call about whether the provider you're currently using is good enough for your use case.

Voy a ofrecer una última advertencia sobre la elección de una plataforma de despliegue y un enfoque general del despliegue. Algunas personas te guiarán con entusiasmo hacia enfoques y servicios de despliegue excesivamente complejos, destinados a hacer que tu proyecto sea altamente fiable y capaz de servir a millones de usuarios simultáneamente. Muchos programadores invierten mucho tiempo, dinero y energía en crear una compleja estrategia de despliegue, sólo para descubrir que casi nadie utiliza su proyecto. La mayoría de los proyectos Django pueden configurarse en un pequeño plan de alojamiento y ajustarse para servir miles de peticiones por minuto. Si tu proyecto recibe algo menos que este nivel de tráfico, dedica tiempo a configurar tu despliegue para que funcione bien en una plataforma mínima antes de invertir en una infraestructura pensada para algunos de los sitios más grandes del mundo.

I'll offer one last word of caution about choosing a deployment platform and an overall approach to deployment. Some people will enthusiastically steer you toward overly complex deployment approaches and services that are meant to make your project highly reliable and capable of serving millions of users simultaneously. Many programmers spend lots of time, money, and energy building out a complex deployment strategy, only to find that hardly anyone is using their project. Most Django projects can be set up on a small hosting plan and tuned to serve thousands of requests per minute. If your project is getting anything less than this level of traffic, take the time to configure your deployment to work well on a minimal platform before investing in infrastructure that's meant for some of the largest sites in the world.

El despliegue es increíblemente desafiante a veces, pero igual de satisfactorio cuando tu proyecto en vivo funciona bien. Disfruta del reto, y pide ayuda cuando la necesites.

Deployment is incredibly challenging at times, but just as satisfying when your live project works well. Enjoy the challenge, and ask for help when you need it.

Índice (Index)

Índice (Index)

Ten en cuenta que el índice enlaza con la ubicación aproximada de cada término.

Please note that index links to approximate location of each term.

Símbolos

Symbols

+ (adición), [26](#)

+ (addition), [26](#)

+= (adición en su lugar), [122](#)

+= (addition in place), [122](#)

* (argumentos arbitrarios), [146](#)

* (arbitrary arguments), [146](#)

** (argumentos de palabra clave arbitrarios), [148](#)

****** (arbitrary keyword arguments), [148](#)

{} (corchetes)

{} (braces)

diccionarios, [92](#)

dictionaries, [92](#)

conjuntos, [104](#)

sets, [104](#)

@ (decorador), [221](#) , [424](#)

@ (decorator), [221](#) , [424](#)

/ (división), [26](#)

/ (division), [26](#)

== (igualdad), [72](#) , [74](#)

== (equality), [72](#) , [74](#)

****** (exponente), [26](#)

****** (exponent), [26](#)

> (mayor que), [75](#)

> (greater than), [75](#)

>= (mayor o igual que), [75](#)

>= (greater than or equal to), [75](#)

(almohadilla), para comentarios, [29](#)

(hash mark), for comments, [29](#)

!= (desigualdad), [74](#)

!= (inequality), [74](#)

< (menos de), [75](#)

< (less than), [75](#)

<= (menor o igual que), [75](#)

`<=` (less than or equal to), [75](#)

`[]` (lista), [34](#)

`[]` (list), [34](#)

`%` (modulo), [116](#)

`%` (modulo), [116](#)

`+=` (cadenas multilínea), [115](#)

`+=` (multiline strings), [115](#)

`*` (multiplicación), [26](#)

`*` (multiplication), [26](#)

`\n` (nueva línea), [22](#)

`\n` (newline), [22](#)

`>>>` (Python prompt), [4](#)

`>>>` (Python prompt), [4](#)

- (resta), [26](#)

- (subtraction), [26](#)

\t (pestaña), [22](#)

\t (tab), [22](#)

_ (subrayado)

_ (underscore)

en nombres de archivos y carpetas, [10](#)

in file and folder names, [10](#)

en números, [28](#)

in numbers, [28](#)

en nombres de variables, [17](#)

in variable names, [17](#)

A

A

alias, [151 - 152](#) , [178 - 179](#)

aliases, [151 – 152](#) , [178 – 179](#)

alice.py , [195 - 197](#)

alice.py , [195 – 197](#)

Alien Invasion . *See also* Pygame

Alien Invasion . *See also* Pygame

extraterrestres, [256 - 274](#)

aliens, [256 – 274](#)

construir flota, [259 - 262](#)

building fleet, [259 – 262](#)

comprobando bordes, [265](#)

checking edges, [265](#)

colisiones, con balas [267](#)

collisions, with bullets, [267](#)

colisiones, con nave, [270 - 273](#)

collisions, with ship, [270 - 273](#)

control de la dirección de la flota, [264 - 266](#)

controlling fleet direction, [264 - 266](#)

creando un extraterrestre, [256 - 258](#)

creating an alien, [256 - 258](#)

dejando caer flota, [265 - 266](#)

dropping fleet, [265 - 266](#)

llegando a la parte inferior de la pantalla, [273 - 274](#)

reaching bottom of screen, [273 - 274](#)

reconstruyendo la flota, [268 - 269](#)

rebuilding fleet, [268 – 269](#)

viñetas, [247 - 253](#) , [266 - 270](#)

bullets, [247 – 253](#) , [266 – 270](#)

colisiones, con extraterrestres [267](#)

collisions, with aliens, [267](#)

borrar antiguo, [250 - 251](#)

deleting old, [250 – 251](#)

disparando, [249 - 250](#)

firing, [249 – 250](#)

más grande, [268](#)

larger, [268](#)

número limitado de, [251 - 252](#)

limiting number of, [251 – 252](#)

configuración, [247](#)

settings, [247](#)

acelerando, [269](#)

speeding up, [269](#)

clases

classes

Alien , [257](#)

Alien , [257](#)

AlienInvasion , [229](#)

AlienInvasion , [229](#)

Bullet , [247](#) - [248](#)

Bullet , [247](#) - [248](#)

Button , [278](#) - [279](#)

Button , [278 – 279](#)

GameStats , [271](#)

GameStats , [271](#)

Scoreboard , [286 - 287](#)

Scoreboard , [286 – 287](#)

Settings , [232](#)

Settings , [232](#)

Ship , [234 - 235](#)

Ship , [234 – 235](#)

fin del juego, [274 - 275](#)

ending the game, [274 – 275](#)

inicializando configuraciones dinámicas, [283 - 285](#)

initializing dynamic settings, [283 – 285](#)

niveles

levels

modificando los ajustes de velocidad, [283 - 285](#)

modifying speed settings, [283 - 285](#)

restablecer la velocidad [285](#)

resetting the speed, [285](#)

visualizando, [294 - 296](#)

displaying, [294 - 296](#)

flota en movimiento, [263 - 266](#)

moving fleet, [263 - 266](#)

planificación, [228](#)

planning, [228](#)

Botón de reproducción, [278 - 283](#)

Play button, [278 – 283](#)

Button clase, [278 - 279](#)

Button class, [278 – 279](#)

desactivar, [282](#)

deactivating, [282](#)

dibujo, [279 - 280](#)

drawing, [279 – 280](#)

ocultando el cursor del ratón, [282 - 283](#)

hiding mouse cursor, [282 – 283](#)

reiniciando el juego, [281 - 282](#)

resetting game, [281 – 282](#)

juego inicial, [281](#)

starting game, [281](#)

revisando el proyecto, [256](#)

reviewing the project, [256](#)

puntuación, [286 - 298](#)

scoring, [286 - 298](#)

todos los hits, [290](#)

all hits, [290](#)

puntuación alta, [292 - 294](#)

high score, [292 - 294](#)

aumentando los valores de los puntos, [290 - 291](#)

increasing point values, [290 - 291](#)

nivel, [294 - 296](#)

level, [294 - 296](#)

número de naves, [296 - 299](#)

number of ships, [296 – 299](#)

reiniciar, [289 - 290](#)

resetting, [289 – 290](#)

redondeo y formato, [291 - 292](#)

rounding and formatting, [291 – 292](#)

atributo puntuación, [286](#)

score attribute, [286](#)

actualización, [289](#)

updating, [289](#)

configuración, almacenamiento, [232 - 233](#)

settings, storing, [232 – 233](#)

nave, [233 - 244](#)

ship, [233 – 244](#)

ajustando la velocidad, [242 - 243](#)

adjusting speed, [242 – 243](#)

movimiento continuo, [239 - 242](#)

continuous movement, [239 – 242](#)

encontrar una imagen, [233 - 234](#)

finding an image, [233 – 234](#)

alcance limitado, [243 - 244](#)

limiting range, [243 – 244](#)

amusement_park.py, [80 - 82](#)

amusement_park.py, [80 – 82](#)

and palabra clave, [75](#)

and keyword, [75](#)

antialiasing, [279](#)

antialiasing, [279](#)

API. *See* interfaz de programación de aplicaciones

API. *See* application programming interface

apostrophe.py, [24 - 25](#)

apostrophe.py, [24 – 25](#)

append() método, [37 - 38](#)

append() method, [37 – 38](#)

interfaz de programación de aplicaciones (API), [355](#)

application programming interface (API), [355](#)

Llamada a la API, [355 - 357](#)

API call, [355 – 357](#)

API de GitHub, [368](#)

GitHub API, [368](#)

API de Hacker News, [368 - 371](#)

Hacker News API, [368 – 371](#)

procesando una respuesta API, [357 - 362](#)

processing an API response, [357 – 362](#)

límites de velocidad, [362](#)

rate limits, [362](#)

solicitando datos, [356 - 357](#)

requesting data, [356 – 357](#)

visualización de resultados, [362 - 368](#)

visualizing results, [362 – 368](#)

argumentos, [131](#) . *See also under* funciones

arguments, [131](#) . *See also under* functions

as palabra clave, [151 - 152](#)

as keyword, [151](#) – [152](#)

aserciones, [213](#) , [217](#) - [218](#)

assertions, [213](#) , [217](#) – [218](#)

atributos, [159](#) . *See also under* classes

attributes, [159](#) . *See also under* classes

B

B

banned_users.py , [76](#) - [77](#)

banned_users.py , [76](#) – [77](#)

bicycles.py , [34](#) - [35](#)

bicycles.py , [34](#) – [35](#)

Valores booleanos, [77](#)

Boolean values, [77](#)

Bootstrap, [433](#) . *See also under* Django

Bootstrap, [433](#) . *See also under* Django

llaves ({})

braces ({})

diccionarios, [92](#)

dictionaries, [92](#)

conjuntos, [104](#)

sets, [104](#)

break declaración, [121](#)

break statement, [121](#)

funciones incorporadas, [467](#)

built-in functions, [467](#)

c

c

llamadas (funciones), [130](#) , [132 - 135](#)

calls (functions), [130](#) , [132 - 135](#)

car.py , [162 - 178](#)

car.py , [162 - 178](#)

cars.py , [43 - 45](#) , [72](#)

cars.py , [43 - 45](#) , [72](#)

cities.py , [121](#)

cities.py , [121](#)

clases

classes

atributos, [159](#)

attributes, [159](#)

acceder, [160](#)

accessing, [160](#)

valores por defecto, [163](#) - [164](#)

default values, [163](#) – [164](#)

modificando, [164](#) - [166](#)

modifying, [164](#) – [166](#)

crear, [158](#) - [161](#)

creating, [158](#) – [161](#)

importando, [173](#) - [179](#)

importing, [173](#) – [179](#)

varias clases, [175](#) - [176](#)

multiple classes, [175](#) – [176](#)

clases individuales, [174](#) - [175](#)

single classes, [174 – 175](#)

herencia, [167 - 172](#)

inheritance, [167 – 172](#)

atributos y métodos, [169](#)

attributes and methods, [169](#)

clases para niños, [167 - 170](#)

child classes, [167 – 170](#)

composición, [170](#)

composition, [170](#)

`__init__()` método, [167 - 169](#)

`__init__()` method, [167 – 169](#)

instancias como atributos, [170 - 172](#)

instances as attributes, [170 – 172](#)

anular métodos, [170](#)

overriding methods, [170](#)

clases principales, [170](#)

parent classes, [170](#)

subclases, [168](#)

subclasses, [168](#)

super() función, [168](#)

super() function, [168](#)

superclases, [168](#)

superclasses, [168](#)

instancias, [157](#)

instances, [157](#)

métodos, [159](#)

methods, [159](#)

llamando, [160](#)

calling, [160](#)

encadenamiento, [185](#)

chaining, [185](#)

`__init()` método, [159](#)

`__init()` method, [159](#)

modelado de objetos del mundo real, [172 - 173](#)

modeling real-world objects, [172 - 173](#)

múltiples instancias, [161](#)

multiple instances, [161](#)

convenciones de nomenclatura, [158](#)

naming conventions, [158](#)

objetos, [157](#)

objects, [157](#)

normas de estilo, [181](#)

style guidelines, [181](#)

Archivos de valores separados por comas. *See* Archivos CSV

comma-separated value files. *See* CSV files

comment.py , [29](#)

comment.py , [29](#)

comentarios, [29](#) - [30](#)

comments, [29](#) – [30](#)

pruebas condicionales, [72](#) - [77](#) . *See also* if declaraciones

conditional tests, [72](#) – [77](#) . *See also* if statements

confirmed_users.py , [124](#) - [125](#)

confirmed_users.py , [124](#) – [125](#)

constantes, [28](#)

constants, [28](#)

continue declaración, [122](#)

continue statement, [122](#)

counting.py , [117](#) - [118](#) , [122](#) - [123](#)

counting.py , [117](#) – [118](#) , [122](#) – [123](#)

Archivos CSV, [330](#) - [341](#)

CSV files, [330](#) – [341](#)

csv.reader() función, [330](#) - [333](#)

csv.reader() function, [330](#) – [333](#)

comprobación de errores, [338](#) - [341](#)

error checking, [338](#) – [341](#)

cabeceras de archivo, [330 - 332](#)

file headers, [330 – 332](#)

D

D

análisis de datos, [301](#)

data analysis, [301](#)

bases de datos. *See under* Django

databases. *See under* Django

visualización de datos, [301](#) . *See also* Matplotlib; Plotly

data visualization, [301](#) . *See also* Matplotlib; Plotly

datetime módulo, [333 - 335](#)

datetime module, [333 – 335](#)

death_valley_highs_lows.py , [339 - 341](#)

death_valley_highs_lows.py, [339 – 341](#)

decoradores, [221 - 223](#), [423 - 425](#)

decorators, [221 – 223](#), [423 – 425](#)

valores por defecto

default values

atributos de clase, [163 - 164](#)

class attributes, [163 – 164](#)

parámetros de función, [134 - 135](#)

function parameters, [134 – 135](#)

definición (funciones), [130](#)

definition (functions), [130](#)

def palabra clave, [130](#)

def keyword, [130](#)

del declaración

del statement

con diccionarios, [96](#)

with dictionaries, [96](#)

con listas, [38 - 40](#)

with lists, [38 - 40](#)

dice_visual_d6d10.py, [326 - 327](#)

dice_visual_d6d10.py, [326 - 327](#)

dice_visual.py, [324 - 326](#)

dice_visual.py, [324 - 326](#)

diccionarios

dictionaries

definir, [92](#)

defining, [92](#)

vacío, [94](#)

empty, [94](#)

formato más grande, [96](#) - [97](#)

formatting larger, [96](#) – [97](#)

KeyError , [98](#)

KeyError , [98](#)

pares clave-valor, [92](#)

key-value pairs, [92](#)

añadiendo, [93](#) - [94](#)

adding, [93](#) – [94](#)

eliminar, [96](#)

removing, [96](#)

bucle a través de

looping through

teclas, [101 - 102](#)

keys, [101 – 102](#)

teclas en orden, [102 - 103](#)

keys in order, [102 – 103](#)

pares clave-valor, [99 - 101](#)

key-value pairs, [99 – 101](#)

valores, [103 - 104](#)

values, [103 – 104](#)

métodos

methods

get() , [97 - 98](#)

`get()` , [97 – 98](#)

`items()` , [99 - 101](#)

`items()` , [99 – 101](#)

`keys()` , [101 - 103](#)

`keys()` , [101 – 103](#)

`values()` , [103 - 104](#)

`values()` , [103 – 104](#)

anidamiento

nesting

diccionarios en diccionarios, [110 - 111](#)

dictionaries in dictionaries, [110 – 111](#)

diccionarios en listas, [105 - 108](#)

dictionaries in lists, [105 – 108](#)

listas en diccionarios, [108 - 109](#)

lists in dictionaries, [108 – 109](#)

ordenando en, [94](#) , [102 - 103](#)

ordering in, [94](#) , [102 – 103](#)

ordenar una lista de, [370](#)

sorting a list of, [370](#)

valores

values

acceder, [92 - 93](#) , [97 - 98](#)

accessing, [92 – 93](#) , [97 – 98](#)

modificando, [94 - 96](#)

modifying, [94 – 96](#)

die.py , [320](#)

die.py , [320](#)

die_visual.py , [320 - 321](#)

die_visual.py , [320 - 321](#)

dimensions.py , [66 - 67](#)

dimensions.py , [66 - 67](#)

div (HTML), [437](#)

div (HTML), [437](#)

division_calculator.py , [192 - 195](#)

division_calculator.py , [192 - 195](#)

Django. *See also* [Git](#); [Proyecto Learning Log](#)

Django. *See also* [Git](#); [Learning Log project](#)

aplicación de cuentas, [415 - 423](#)

accounts app, [415 - 423](#)

creación de aplicaciones, [415 - 416](#)

creating app, [415 - 416](#)

cerrar sesión, [419 - 420](#)

logging out, [419 - 420](#)

página de acceso, [416 - 419](#)

login page, [416 - 419](#)

página de inscripción, [420 - 423](#)

registration page, [420 - 423](#)

admin site, [381 - 386](#)

admin site, [381 - 386](#)

asociar datos a un usuario, [425 - 430](#)

associating data with a user, [425 - 430](#)

Bootstrap, [434 - 445](#)

Bootstrap, [434 – 445](#)

tarjeta, [443](#)

card, [443](#)

navegación plegable, [437](#)

collapsible navigation, [437](#)

contenedor, [440](#)

container, [440](#)

aplicación django-bootstrap5, [434](#)

django-bootstrap5 app, [434](#)

documentación, [444](#)

documentation, [444](#)

Cabeceras HTML, [435 - 436](#)

HTML headers, [435 – 436](#)

jumbotron, [440 - 441](#)

jumbotron, [440 – 441](#)

grupos de la lista, [443](#)

list groups, [443](#)

barra de navegación, [436 - 439](#)

navigation bar, [436 – 439](#)

estilizando formularios, [441 - 442](#)

styling forms, [441 – 442](#)

comandos

commands

createsuperuser , [382](#)

createsuperuser , [382](#)

flush , [427](#)

flush , [427](#)

makemigrations , [381](#) , [385](#) , [426](#)

makemigrations , [381](#) , [385](#) , [426](#)

migrate , [377](#)

migrate , [377](#)

runserver , [377 - 378](#) , [383](#) , [392](#)

runserver , [377 - 378](#) , [383](#) , [392](#)

shell , [386](#)

shell , [386](#)

startapp , [379](#) , [415](#)

startapp , [379](#) , [415](#)

startproject , [376](#)

startproject , [376](#)

crear nuevos proyectos, [376](#)

creating new projects, [376](#)

bases de datos

databases

borrar en cascada, [384](#)

cascading delete, [384](#)

crear, [376](#)

creating, [376](#)

claves externas, [384](#) , [425](#)

foreign keys, [384](#) , [425](#)

relaciones muchos-a-uno, [384](#)

many-to-one relationships, [384](#)

migrando, [377](#) , [381](#) , [385](#) , [426](#)

migrating, [377](#) , [381](#) , [385](#) , [426](#)

campo no anulable, [427](#)

non-nullable field, [427](#)

Postgres, [447](#)

Postgres, [447](#)

consultas, [398](#) , [428](#)

queries, [398](#) , [428](#)

querysets, [386](#) - [387](#) , [395](#) , [398](#) , [426](#) - [428](#)

querysets, [386](#) – [387](#) , [395](#) , [398](#) , [426](#) – [428](#)

reiniciar, [427](#)

resetting, [427](#)

SQLite, [377](#)

SQLite, [377](#)

despliegue, [445 - 461](#) , [493 - 501](#)

deployment, [445 - 461](#) , [493 - 501](#)

comprometiendo el proyecto, [453](#)

committing the project, [453](#)

archivos de configuración, [447 - 450](#)

configuration files, [447 - 450](#)

creando el proyecto Platform.sh, [453 - 455](#)

creating Platform.sh project, [453 - 455](#)

crear superusuario, [456 - 457](#)

creating superuser, [456 - 457](#)

páginas de error personalizadas, [459 - 460](#)

custom error pages, [459 - 460](#)

borrando proyectos, [461](#)

deleting projects, [461](#)

límites de la prueba gratuita, [446](#)

free trial limits, [446](#)

gunicorn , [447](#)

gunicorn , [447](#)

ignorando archivos, [452](#) - [453](#)

ignoring files, [452](#) – [453](#)

instalando Platform.sh CLI, [446](#) , [497](#) - [500](#)

installing Platform.sh CLI, [446](#) , [497](#) – [500](#)

instalando platformshconfig , [446](#)

installing platformshconfig , [446](#)

otros enfoques de despliegue, [500](#)

other deployment approaches, [500](#)

Plataforma.sh, [445](#)

Platform.sh, [445](#)

Base de datos Postgres, [447](#) , [450](#) - [451](#)

Postgres database, [447](#) , [450](#) – [451](#)

psycopg2 , [447](#)

psycopg2 , [447](#)

impulsando un proyecto, [455](#)

pushing a project, [455](#)

enviando cambios, [458](#) , [460](#)

pushing changes, [458](#) , [460](#)

requirements.txt , [446](#)

requirements.txt , [446](#)

proyecto de seguridad, [457](#) - [460](#)

securing project, [457 – 460](#)

configuración, [451](#)

settings, [451](#)

Sesiones SSH, [456 - 457](#)

SSH sessions, [456 – 457](#)

resolución de problemas, [494 - 501](#)

troubleshooting, [494 – 501](#)

utilizando Git, [451](#)

using Git, [451](#)

ver proyecto, [456](#)

viewing project, [456](#)

servidor de desarrollo, [377 - 378](#) , [383](#) , [392](#)

development server, [377 – 378](#) , [383](#) , [392](#)

documentación

documentation

campos del modelo, [380](#)

model fields, [380](#)

consultas, [388](#)

queries, [388](#)

plantillas, [400](#)

templates, [400](#)

formularios, [404 - 423](#) , [429 - 430](#)

forms, [404 - 423](#) , [429 - 430](#)

csrf_token , [407](#)

csrf_token , [407](#)

Peticiones GET y POST, [406](#)

GET and POST requests, [406](#)

ModelForm , [404](#) , [408](#)

ModelForm , [404](#) , [408](#)

procesando formularios, [405 - 406](#) , [409 - 410](#) , [412 - 413](#) , [421 - 422](#) , [429 - 430](#)

processing forms, [405 - 406](#) , [409 - 410](#) , [412 - 413](#) , [421 - 422](#) , [429 - 430](#)

save() método, [405 - 406](#) , [409 - 410](#) , [430](#)

save() method, [405 - 406](#) , [409 - 410](#) , [430](#)

plantillas, [407](#) , [410 - 411](#) , [413](#) , [417](#) , [419](#) , [422](#)

templates, [407](#) , [410 - 411](#) , [413](#) , [417](#) , [419](#) , [422](#)

validación, [404 - 406](#)

validation, [404 - 406](#)

widgets, [408](#)

widgets, [408](#)

HTML

HTML

etiqueta de anclaje (`<a>`), [393](#)

anchor tag (`<a>`), [393](#)

`<body>` elemento, [437](#)

`<body>` element, [437](#)

comentarios, [437](#)

comments, [437](#)

`<div>` elementos, [437](#)

`<div>` elements, [437](#)

`<main>` elemento, [440](#)

`<main>` element, [440](#)

márgenes, [440](#)

margins, [440](#)

relleno, [440](#)

padding, [440](#)

<p> elementos, [391](#)

<p> elements, [391](#)

 elementos, [438](#)

 elements, [438](#)

Error HTTP 404, [428 - 429](#) , [459 - 460](#)

HTTP 404 error, [428 – 429](#) , [459 – 460](#)

INSTALLED_APPS , [380](#)

INSTALLED_APPS , [380](#)

instalación, [375 - 376](#)

installing, [375 – 376](#)

localhost, [378](#)

localhost, [378](#)

cerrar sesión, [419 - 420](#)

logging out, [419 - 420](#)

@login_required decorador, [423 - 424](#)

@login_required decorator, [423 - 424](#)

plantilla de acceso, [417](#)

login template, [417](#)

mapeo de URLs, [388 - 390](#) , [397 - 398](#)

mapping URLs, [388 - 390](#) , [397 - 398](#)

migrando la base de datos, [426 - 427](#)

migrating the database, [426 - 427](#)

modelos, [379](#)

models, [379](#)

activando, [380 - 381](#)

activating, [380 – 381](#)

definiendo, [379](#) , [384](#)

defining, [379](#) , [384](#)

claves externas, [384](#) , [425](#)

foreign keys, [384](#) , [425](#)

registrarse con admin, [382 - 383](#) , [385 - 386](#)

registering with admin, [382 – 383](#) , [385 – 386](#)

`__str__()` método, [380](#) , [384](#)

`__str__()` method, [380](#) , [384](#)

proyectos (vs. aplicaciones), [379](#)

projects (vs. apps), [379](#)

redirect() función, [405 - 406](#)

redirect() function, [405 – 406](#)

ciclo de publicación, [376](#)

release cycle, [376](#)

restringiendo el acceso a los datos, [427 - 430](#)

restricting access to data, [427 – 430](#)

configuración

settings

ALLOWED_HOSTS , [451](#)

ALLOWED_HOSTS , [451](#)

DEBUG , [457 - 458](#)

DEBUG , [457 – 458](#)

INSTALLED_APPS , [380 - 381](#) , [415 - 416](#) , [434](#)

INSTALLED_APPS , [380 – 381](#) , [415 – 416](#) , [434](#)

LOGIN_REDIRECT_URL , [417 - 418](#)

LOGIN_REDIRECT_URL , [417 – 418](#)

LOGIN_URL , [424](#)

LOGIN_URL , [424](#)

LOGOUT_REDIRECT_URL , [420](#)

LOGOUT_REDIRECT_URL , [420](#)

SECRET_KEY , [451](#)

SECRET_KEY , [451](#)

shell, [386 - 387](#) , [426 - 427](#)

shell, [386 – 387](#) , [426 – 427](#)

iniciar una aplicación, [379](#)

starting an app, [379](#)

estilismo. *See* Django: Bootstrap

styling. *See* Django: Bootstrap

superusuarios, [382](#) , [456](#) - [457](#)

superusers, [382](#) , [456](#) – [457](#)

plantillas

templates

etiquetas de bloque, [393](#)

block tags, [393](#)

plantilla hija, [393](#) - [394](#)

child template, [393](#) – [394](#)

diccionario contextual, [395](#)

context dictionary, [395](#)

filtros, [399](#)

filters, [399](#)

forma en, [407](#)

forms in, [407](#)

sangría en, [393](#)

indentation in, [393](#)

herencia, [392](#) - [394](#)

inheritance, [392](#) – [394](#)

linebreaks , [399](#)

linebreaks , [399](#)

enlaces en, [392](#) - [393](#) , [399](#)

links in, [392](#) – [393](#) , [399](#)

bucles en, [395](#) - [397](#)

loops in, [395](#) – [397](#)

plantilla padre, [392 - 393](#)

parent template, [392 – 393](#)

etiquetas de plantilla, [393](#)

template tags, [393](#)

marcas de tiempo en, [398 - 399](#)

timestamps in, [398 – 399](#)

user objeto, [418](#)

user object, [418](#)

escritura, [390 - 392](#)

writing, [390 – 392](#)

URLs. *See* Django: mapeo de URLs

URLs. *See* Django: mapping URLs

UserCreationForm , [421 - 422](#)

UserCreationForm , [421](#) – [422](#)

valores de ID de usuario, [426](#)

user ID values, [426](#)

versiones, [376](#)

versions, [376](#)

ver funciones, [388](#) , [390](#)

view functions, [388](#) , [390](#)

entornos virtuales, [374](#) - [375](#)

virtual environments, [374](#) – [375](#)

docstrings, [130](#) , [153](#) , [181](#)

docstrings, [130](#) , [153](#) , [181](#)

dog.py , [158](#) - [162](#)

dog.py , [158](#) – [162](#)

notación por puntos, [150](#) , [160](#)

dot notation, [150](#) , [160](#)

E

E

terremotos. *See* mapeo de terremotos

earthquakes. *See* mapping earthquakes

electric_car.py , [167](#) - [173](#)

electric_car.py , [167](#) – [173](#)

módulo, [177](#) - [179](#)

module, [177](#) – [179](#)

encoding argumento, [195](#) - [196](#)

encoding argument, [195](#) – [196](#)

enumerate() función, [331](#)

`enumerate()` function, [331](#)

`eq_explore_data.py` , [343](#) - [347](#)

`eq_explore_data.py` , [343](#) – [347](#)

operador de igualdad (`==`), [72](#) , [74](#)

equality operator (`==`), [72](#) , [74](#)

`eq_world_map.py` , [347](#) - [352](#)

`eq_world_map.py` , [347](#) – [352](#)

`even_numbers.py` , [58](#)

`even_numbers.py` , [58](#)

`even_or_odd.py` , [117](#)

`even_or_odd.py` , [117](#)

excepciones, [183](#) , [192](#) - [199](#)

exceptions, [183](#) , [192](#) – [199](#)

Decidir de qué errores informar, [199](#)

deciding which errors to report, [199](#)

else bloque, [194 - 195](#)

else block, [194 - 195](#)

fallando silenciosamente, [198 - 199](#)

failing silently, [198 - 199](#)

FileNotFoundException error, [195 - 196](#)

FileNotFoundException error, [195 - 196](#)

manejo de excepciones, [192 - 196](#)

handling exceptions, [192 - 196](#)

prevención de fallos, [193 - 195](#)

preventing crashes, [193 - 195](#)

try-except bloques, [193](#)

try-except blocks, [193](#)

ZeroDivisionError , [192 - 195](#)

ZeroDivisionError , [192 - 195](#)

exponentes (**), [26](#)

exponents (**), [26](#)

F

F

favorite_languages.py , [96 - 97](#) , [100 - 104](#) , [109](#)

favorite_languages.py , [96 - 97](#) , [100 - 104](#) , [109](#)

file_reader.py , [184 - 187](#)

file_reader.py , [184 - 187](#)

archivos

files

encoding argumento, [195 - 196](#)

encoding argument, [195 – 196](#)

FileNotFound error, [195 - 196](#)

FileNotFound error, [195 – 196](#)

rutas de archivos, [186](#)

file paths, [186](#)

absoluto, [186](#)

absolute, [186](#)

exists() método, [203 - 204](#)

exists() method, [203 – 204](#)

pathlib módulo, [184](#)

pathlib module, [184](#)

Path objetos, [184 - 186](#) , [330](#)

Path objects, [184 – 186](#) , [330](#)

pariente, [186](#)

relative, [186](#)

de cadenas, [198](#)

from strings, [198](#)

en Windows, [186](#)

on Windows, [186](#)

read_text() método, [185](#) , [195 – 196](#)

read_text() method, [185](#) , [195 – 196](#)

splitlines() método, [186 – 187](#)

splitlines() method, [186 – 187](#)

write_text() método, [190 – 191](#)

write_text() method, [190 – 191](#)

first_numbers.py , [57](#)

first_numbers.py , [57](#)

fixtures, [221 - 223](#)

fixtures, [221 - 223](#)

banderas, [120 - 121](#)

flags, [120 - 121](#)

flotantes, [26 - 28](#)

floats, [26 - 28](#)

foods.py , [63 - 64](#)

foods.py , [63 - 64](#)

for bucles, [49 - 56](#) , [99 - 104](#) . *See also* diccionarios; listas

for loops, [49 - 56](#) , [99 - 104](#) . *See also* dictionaries; lists

formatted_name.py , [137 - 139](#)

formatted_name.py , [137](#) – [139](#)

cadenas-f

f-strings

especificadores de formato, [291](#) - [292](#)

format specifiers, [291](#) – [292](#)

usando variables en, [20](#) - [21](#)

using variables in, [20](#) – [21](#)

full_name.py , [21](#)

full_name.py , [21](#)

funciones, [129](#) - [155](#)

functions, [129](#) – [155](#)

argumentos

arguments

arbitrario, [146 - 149](#)

arbitrary, [146 - 149](#)

valores por defecto, [134 - 135](#)

default values, [134 - 135](#)

errores, [136](#)

errors, [136](#)

palabra clave, [133 - 134](#)

keyword, [133 - 134](#)

listas como, [142 - 145](#)

lists as, [142 - 145](#)

opcional, [138 - 139](#)

optional, [138 - 139](#)

posicional, [131 - 133](#)

positional, [131 – 133](#)

cuerpo, [130](#)

body, [130](#)

incorporado, [467](#)

built-in, [467](#)

llamando a funciones, [130](#) , [132 - 135](#)

calling functions, [130](#) , [132 – 135](#)

definir, [130](#)

defining, [130](#)

importando, [149 - 153](#)

importing, [149 – 153](#)

alias, [151 - 152](#)

aliases, [151 – 152](#)

módulos completos, [150 - 151](#)

entire modules, [150 – 151](#)

funciones específicas, [151](#)

specific functions, [151](#)

modificando una lista en una función, [142 - 145](#)

modifying a list in a function, [142 – 145](#)

módulos, [149 - 153](#)

modules, [149 – 153](#)

parámetros, [131](#)

parameters, [131](#)

valores de retorno, [137 - 141](#)

return values, [137 – 141](#)

normas de estilo, [153](#)

style guidelines, [153](#)

G

G

Archivos GeoJSON, [342 - 347](#) , [350 - 351](#)

GeoJSON files, [342 – 347](#) , [350 – 351](#)

Peticiones GET, [406](#) . *See* Django: formularios

GET requests, [406](#) . *See* Django: forms

obtener ayuda

getting help

Discord, [480](#)

Discord, [480](#)

documentación oficial de Python, [479 - 480](#)

official Python documentation, [479 – 480](#)

recursos en línea, [xxxv](#) , [478](#)

online resources, [xxxv](#) , [478](#)

r/learnpython, [480](#)

r/learnpython, [480](#)

depuración con patito de goma, [478](#)

rubber duck debugging, [478](#)

buscando en Internet, [479](#)

searching online, [479](#)

Slack, [481](#)

Slack, [481](#)

Desbordamiento de pila, [479](#)

Stack Overflow, [479](#)

tres preguntas principales, [477](#) - [478](#)

three main questions, [477 – 478](#)

Git, [356](#) , [451 – 453](#) , [483 – 492](#) . *See also* Django: despliegue

Git, [356](#) , [451 – 453](#) , [483 – 492](#) . *See also* Django: deployment

abandonando los cambios, [488 – 489](#)

abandoning changes, [488 – 489](#)

añadiendo archivos, [486](#)

adding files, [486](#)

ramas, [486](#)

branches, [486](#)

consultar commits anteriores, [489 – 491](#)

checking out previous commits, [489 – 491](#)

commits, [486 – 488](#)

commits, [486 – 488](#)

configurando, [452](#) , [484](#)

configuring, [452](#) , [484](#)

borrando un repositorio, [491](#) - [492](#)

deleting a repository, [491](#) – [492](#)

.gitignore , [484](#)

.gitignore , [484](#)

HEAD, [490](#)

HEAD, [490](#)

ignorando archivos, [484](#)

ignoring files, [484](#)

inicializando un repositorio, [485](#)

initializing a repository, [485](#)

instalar, [484](#)

installing, [484](#)

log, [487](#)

log, [487](#)

repositorios, [356](#)

repositories, [356](#)

estado, [485](#) - [486](#)

status, [485](#) – [486](#)

GitHub, [356](#)

GitHub, [356](#)

greeter.py, [114](#) - [115](#), [130](#) - [131](#)

greeter.py, [114](#) – [115](#), [130](#) – [131](#)

greet_users.py, [142](#)

greet_users.py, [142](#)

H

H

API de Hacker News, [368](#) - [371](#)

Hacker News API, [368](#) – [371](#)

almohadilla (#), para comentarios, [29](#)

hash mark (#), for comments, [29](#)

hello_git.py, [484](#) - [491](#)

hello_git.py, [484](#) – [491](#)

hello_world.py, [10](#) - [12](#), [15](#) - [19](#)

hello_world.py, [10](#) – [12](#), [15](#) – [19](#)

archivos ocultos, [448](#), [485](#)

hidden files, [448](#), [485](#)

hn_article.py, [368](#) - [369](#)

hn_article.py, [368 – 369](#)

hn_submissions.py, [369 - 371](#)

hn_submissions.py, [369 – 371](#)

I

I

IDE (entorno de desarrollo integrado), [469 - 470](#)

IDE (integrated development environment), [469 – 470](#)

if declaraciones

if statements

and palabra clave, [75](#)

and keyword, [75](#)

Expresiones booleanas, [77](#)

Boolean expressions, [77](#)

comprobando

checking for

igualdad (==), [72](#)

equality (==), [72](#)

desigualdad (!=), [74](#)

inequality (!=), [74](#)

elemento de la lista, [76](#)

item in list, [76](#)

elemento no en la lista, [76](#)

item not in list, [76](#)

lista no vacía, [86](#) - [87](#)

list not empty, [86](#) – [87](#)

elif declaración, [80](#) - [83](#)

elif statement, [80 – 83](#)

else declaración, [79 - 80](#)

else statement, [79 – 80](#)

if sentencias y listas, [85 - 88](#)

if statements and lists, [85 – 88](#)

ignorando el caso, [73 - 74](#)

ignoring case, [73 – 74](#)

comparaciones numéricas, [74 - 76](#)

numerical comparisons, [74 – 76](#)

or palabra clave, [76](#)

or keyword, [76](#)

sencillo, [78](#)

simple, [78](#)

normas de estilo, [89](#)

style guidelines, [89](#)

comprobación de condiciones múltiples, [82 - 83](#)

testing multiple conditions, [82 – 83](#)

immutable, [65](#)

immutable, [65](#)

import * , [152](#) , [177](#)

import * , [152](#) , [177](#)

import this , [30 - 31](#)

import this , [30 – 31](#)

errores de sangría, [53 - 56](#)

indentation errors, [53 – 56](#)

errores de índice, [46 - 47](#)

index errors, [46](#) – [47](#)

herencia, [167](#) - [173](#) . *See also under* classes

inheritance, [167](#) – [173](#) . *See also under* classes

input() función, [114](#) - [116](#)

input() function, [114](#) – [116](#)

entrada numérica, [115](#) - [116](#)

numerical input, [115](#) – [116](#)

sugerencias de escritura, [114](#) - [115](#)

writing prompts, [114](#) – [115](#)

insert() método, [38](#)

insert() method, [38](#)

itemgetter() función, [370](#)

itemgetter() function, [370](#)

items() método, [99 - 101](#)

items() method, [99 - 101](#)

J

J

Archivos JSON

JSON files

Archivos GeoJSON, [342 - 347](#) , [350 - 351](#)

GeoJSON files, [342 - 347](#) , [350 - 351](#)

Formato de datos JSON, [201](#)

JSON data format, [201](#)

json.dumps() función, [201 - 204](#) , [343 - 344](#) , [368](#)

json.dumps() function, [201 - 204](#) , [343 - 344](#) , [368](#)

json.loads() función, [201 - 204](#) , [343 - 344](#)

`json.loads()` function, [201 – 204](#), [343 – 344](#)

K

K

`keys()` método, [101 - 103](#)

`keys()` method, [101 – 103](#)

pares clave-valor, [92](#) . *See also* diccionarios

key-value pairs, [92](#) . *See also* dictionaries

argumentos de palabras clave, [133 - 134](#)

keyword arguments, [133 – 134](#)

palabras clave, [466](#)

keywords, [466](#)

L

L

language_survey.py , [219](#)

language_survey.py , [219](#)

Proyecto Learning Log, [373](#)

Learning Log project, [373](#)

archivos, [392](#)

files, [392](#)

404.xhtml , [459](#)

404.xhtml , [459](#)

500.xhtml , [459](#)

500.xhtml , [459](#)

accounts/urls.py , [416](#) , [420](#)

accounts/urls.py , [416](#) , [420](#)

accounts/views.py , [421](#) - [422](#)

accounts/views.py , [421](#) – [422](#)

admin.py , [382](#) - [383](#)

admin.py , [382](#) – [383](#)

base.xhtml , [392](#) - [393](#) , [396](#) , [418](#) - [419](#) , [422](#) , [435](#) - [440](#)

base.xhtml , [392](#) – [393](#) , [396](#) , [418](#) – [419](#) , [422](#) , [435](#) – [440](#)

edit_entry.xhtml , [413](#)

edit_entry.xhtml , [413](#)

forms.py , [404](#) , [408](#) - [409](#)

forms.py , [404](#) , [408](#) – [409](#)

.gitignore , [452](#) - [453](#)

.gitignore , [452](#) – [453](#)

index.xhtml , [390](#) - [394](#) , [440](#) - [441](#)

index.xhtml , [390](#) – [394](#) , [440](#) – [441](#)

learning_logs/urls.py , 389 - 390 , 394 - 395 , 397 - 398 , 405 , 409 , 412

learning_logs/urls.py , 389 – 390 , 394 – 395 , 397 – 398 , 405 , 409 , 412

learning_logs/views.py , 390 , 395 , 398 , 405 - 406 , 409 - 410 , 412 - 413 , 423 - 425 , 428 - 430

learning_logs/views.py , 390 , 395 , 398 , 405 – 406 , 409 – 410 , 412 – 413 , 423 – 425 , 428 – 430

ll_project/urls.py , 388 - 389 , 416

ll_project/urls.py , 388 – 389 , 416

login.xhtml , 417 , 441 - 442

login.xhtml , 417 , 441 – 442

models.py , 379 - 380 , 384

models.py , 379 – 380 , 384

new_entry.xhtml , 410

new_entry.xhtml , 410

new_topic.xhtml , [407](#)

new_topic.xhtml , [407](#)

.platform.app.yaml , [448 - 450](#)

.platform.app.yaml , [448 - 450](#)

register.xhtml , [422](#)

register.xhtml , [422](#)

requirements.txt , [446 - 447](#)

requirements.txt , [446 - 447](#)

routes.yaml , [450](#)

routes.yaml , [450](#)

services.yaml , [450](#)

services.yaml , [450](#)

settings.py , [380 - 381](#) , [415 - 418](#) , [420](#) , [424](#) , [434](#) , [451](#) , [457 - 460](#)

settings.py , [380 – 381](#) , [415 – 418](#) , [420](#) , [424](#) , [434](#) , [451](#) , [457 – 460](#)

topic.xhtml , [398 - 399](#) , [443 - 444](#)

topic.xhtml , [398 – 399](#) , [443 – 444](#)

topics.xhtml , [395 - 396](#) , [442 - 443](#)

topics.xhtml , [395 – 396](#) , [442 – 443](#)

desarrollo continuo, [460](#)

ongoing development, [460](#)

páginas, [391](#)

pages, [391](#)

editar entrada, [412 - 414](#)

edit entry, [412 – 414](#)

página de inicio, [388 - 394](#)

home page, [388 – 394](#)

página de acceso, [416 - 419](#)

login page, [416 - 419](#)

nueva entrada, [408 - 411](#)

new entry, [408 - 411](#)

nuevo tema, [404 - 408](#)

new topic, [404 - 408](#)

inscripción, [420 - 423](#)

registration, [420 - 423](#)

tema, [397 - 400](#)

topic, [397 - 400](#)

temas, [394 - 397](#)

topics, [394 - 397](#)

escribir una especificación (spec) [374](#)

writing a specification (spec), [374](#)

len() función, [44](#) - [45](#)

len() function, [44](#) - [45](#)

biblioteca, [184](#)

library, [184](#)

Linux

Linux

Python

Python

comprobando la versión instalada, [8](#)

checking installed version, [8](#)

configuración, [8](#) - [12](#) , [465](#) - [466](#)

setting up, [8](#) - [12](#) , [465](#) - [466](#)

terminales

terminals

ejecutando programas desde, [12](#)

running programs from, [12](#)

Iniciar sesión de Python, [9](#)

starting Python session, [9](#)

resolución de problemas de instalación, [10](#)

troubleshooting installation issues, [10](#)

VS Code, instalando, [9](#)

VS Code, installing, [9](#)

listas, [33](#)

lists, [33](#)

como argumentos, [142](#) - [145](#)

as arguments, [142 – 145](#)

comprensiones, [59 - 60](#)

comprehensions, [59 – 60](#)

copiando, [63 - 64](#)

copying, [63 – 64](#)

elementos

elements

acceder, [34](#)

accessing, [34](#)

accediendo al último, [35](#)

accessing last, [35](#)

añadiendo con `append()` , [37 - 38](#)

adding with `append()` , [37 – 38](#)

añadiendo con `insert()` , [38](#)

adding with `insert()` , [38](#)

identificación única, [104](#)

identifying unique, [104](#)

modificando, [36 - 37](#)

modifying, [36 - 37](#)

eliminando con `del` , [38 - 39](#)

removing with `del` , [38 - 39](#)

eliminando con `pop()` , [39 - 40](#)

removing with `pop()` , [39 - 40](#)

eliminando con `remove()` , [40 - 41](#)

removing with `remove()` , [40 - 41](#)

vacío, [37 - 38](#)

empty, [37 – 38](#)

enumerate() función, [331](#)

enumerate() function, [331](#)

errores

errors

sangría, [53 - 56](#)

indentation, [53 – 56](#)

index, [46](#)

index, [46](#)

for bucles, [49 - 56](#)

for loops, [49 – 56](#)

anidado, [108 - 109](#) , [261 - 262](#)

nested, [108 – 109](#) , [261 – 262](#)

índices, [34 - 35](#)

indexes, [34 - 35](#)

índice negativo, [35](#)

negative index, [35](#)

índice cero, [34 - 35](#)

zero index, [34 - 35](#)

len() función, [44 - 45](#)

len() function, [44 - 45](#)

naming, [33 - 34](#)

naming, [33 - 34](#)

anidamiento

nesting

diccionarios en listas, [105 - 108](#)

dictionaries in lists, [105 – 108](#)

listas en diccionarios, [108 - 109](#)

lists in dictionaries, [108 – 109](#)

listas numéricas, [56 - 60](#)

numerical lists, [56 – 60](#)

max() función, [59](#)

max() function, [59](#)

min() función, [59](#)

min() function, [59](#)

range() función, [58 - 59](#)

range() function, [58 – 59](#)

sum() función, [59](#)

sum() function, [59](#)

eliminar todas las apariciones de un valor, [125](#)

removing all occurrences of a value, [125](#)

rodajas, [61 - 62](#)

slices, [61 - 62](#)

ordenar

sorting

reverse() método, [44](#)

reverse() method, [44](#)

sorted() función, [43 - 44](#)

sorted() function, [43 - 44](#)

sort() método, [43](#)

sort() method, [43](#)

corchetes, [34](#)

square brackets, [34](#)

errores lógicos, [54](#)

logical errors, [54](#)

`lstrip()` método, [22 - 23](#)

`lstrip()` method, [22 - 23](#)

M

M

macOS

macOS

.DS_Store archivos, ignorando, [453](#)

.DS_Store files, ignoring, [453](#)

Homebrew gestor de paquetes, [499](#)

Homebrew package manager, [499](#)

Python

Python

comprobando la versión instalada, [7](#)

checking installed version, [7](#)

configuración, [7 - 12](#) , [464 - 465](#)

setting up, [7 - 12](#) , [464 - 465](#)

terminales

terminals

ejecutando programas desde, [12](#)

running programs from, [12](#)

iniciar sesión en Python, [7](#)

starting Python session, [7](#)

resolución de problemas de instalación, [10](#)

troubleshooting installation issues, [10](#)

VS Code, instalando, [8](#)

VS Code, installing, [8](#)

magicians.py, [49](#) - [56](#)

magicians.py, [49](#) – [56](#)

magic_number.py, [74](#)

magic_number.py, [74](#)

making_pizzas.py, [150](#) - [152](#)

making_pizzas.py, [150](#) – [152](#)

cartografía de terremotos, [342](#) - [352](#) . *See also* Plotly

mapping earthquakes, [342](#) – [352](#) . *See also* Plotly

descargar datos, [343](#) , [352](#)

downloading data, [343](#) , [352](#)

Archivos GeoJSON, [342 - 347](#) , [350 - 351](#)

GeoJSON files, [342 – 347](#) , [350 – 351](#)

ordenación latitud-longitud, [345](#)

latitude-longitude ordering, [345](#)

datos de localización, [346 - 347](#)

location data, [346 – 347](#)

magnitudes, [346](#)

magnitudes, [346](#)

mapa del mundo, [347 - 348](#)

world map, [347 – 348](#)

Matplotlib

Matplotlib

ejes

axes

set_aspect() método, [313 - 314](#)

set_aspect() method, [313 - 314](#)

eliminar, [317](#)

removing, [317](#)

ax objetos, [303](#)

ax objects, [303](#)

mapas de colores, [310 - 311](#)

colormaps, [310 - 311](#)

fig objetos, [303](#)

fig objects, [303](#)

figsize argumento, [318](#)

figsize argument, [318](#)

formato de gráficos

formatting plots

alpha argumento, [337 - 338](#)

alpha argument, [337 - 338](#)

estilos incorporados, [306](#)

built-in styles, [306](#)

colores personalizados, [310](#)

custom colors, [310](#)

etiquetas, [303 - 304](#)

labels, [303 - 304](#)

grosor de línea, [303 - 304](#)

line thickness, [303 - 304](#)

tamaño del gráfico, [318](#)

plot size, [318](#)

sombreado, [337](#) - [338](#)

shading, [337](#) – [338](#)

etiquetas de tick, [309](#) - [310](#)

tick labels, [309](#) – [310](#)

galería, [302](#)

gallery, [302](#)

instalar, [302](#)

installing, [302](#)

plot() método, [303](#) - [306](#)

plot() method, [303](#) – [306](#)

pyplot módulo, [302](#) - [303](#)

pyplot module, [302](#) – [303](#)

savefig() método, [311](#)

savefig() method, [311](#)

guardando parcelas, [311](#)

saving plots, [311](#)

scatter() método, [306 - 311](#)

scatter() method, [306 - 311](#)

gráfico lineal simple, [302 - 306](#)

simple line graph, [302 - 306](#)

subplots() función, [303](#)

subplots() function, [303](#)

métodos, [20](#)

methods, [20](#)

métodos de ayuda, [237](#)

helper methods, [237](#)

módulos, [149 - 152](#) , [173 - 179](#) . *See also* clases: importación; funciones: importación

modules, [149 - 152](#) , [173 - 179](#) . *See also* classes: importing; functions: importing

operador módulo (%), [116 - 117](#)

modulo operator (%), [116 - 117](#)

motorcycles.py , [36 - 41](#)

motorcycles.py , [36 - 41](#)

mountain_poll.py , [125 - 126](#)

mountain_poll.py , [125 - 126](#)

mpl_squares.py , [302 - 306](#)

mpl_squares.py , [302 - 306](#)

my_car.py , [174 - 175](#)

my_car.py , [174 - 175](#)

my_cars.py , [176 - 179](#)

my_cars.py , [176 – 179](#)

my_electric_car.py , [176](#)

my_electric_car.py , [176](#)

N

N

errores de nombre, [17 - 18](#)

name errors, [17 – 18](#)

name_function.py , [211 - 217](#)

name_function.py , [211 – 217](#)

name.py , [20](#)

name.py , [20](#)

names.py , [211 - 212](#)

names.py , [211](#) – [212](#)

anidamiento. *See* diccionarios: anidamiento; listas: for bucles

nesting. *See* dictionaries: nesting; lists: for loops

nueva línea (`\n`), [21](#) - [22](#)

newline (`\n`), [21](#) – [22](#)

next() función, [330](#) - [331](#)

next() function, [330](#) – [331](#)

None , [98](#) , [140](#)

None , [98](#) , [140](#)

number_reader.py , [202](#)

number_reader.py , [202](#)

números, [26](#) - [28](#)

numbers, [26](#) – [28](#)

aritmética, [26](#)

arithmetic, [26](#)

constantes, [28](#)

constants, [28](#)

exponentes, [26](#)

exponents, [26](#)

flotantes, [26](#) - [27](#)

floats, [26](#) - [27](#)

formato, [291](#) - [292](#)

formatting, [291](#) - [292](#)

enteros, [26](#)

integers, [26](#)

mezclando enteros y flotantes, [27](#) - [28](#)

mixing integers and floats, [27](#) – [28](#)

orden de las operaciones, [26](#)

order of operations, [26](#)

round() función, [291](#) - [292](#)

round() function, [291](#) – [292](#)

subraya en, [28](#)

underscores in, [28](#)

number_writer.py , [201](#)

number_writer.py , [201](#)

O

O

programación orientada a objetos (POO), [157](#) . *See also* [classes](#)

object-oriented programming (OOP), [157](#) . *See also* [classes](#)

or palabra clave, [76](#) . *See also* if declaraciones

or keyword, [76](#) . *See also* if statements

P

P

pandas, [320](#)

pandas, [320](#)

parámetros, [131](#)

parameters, [131](#)

parrot.py , [114](#) , [118](#) - [121](#)

parrot.py , [114](#) , [118](#) - [121](#)

pass declaración, [198](#) - [199](#)

pass statement, [198](#) - [199](#)

rutas. *See* archivos: rutas de archivos

paths. *See* files: file paths

PEP 8, [68](#) - [69](#)

PEP 8, [68](#) – [69](#)

person.py , [139](#) - [140](#)

person.py , [139](#) – [140](#)

pets.py , [125](#) , [132](#) - [136](#)

pets.py , [125](#) , [132](#) – [136](#)

pip, [210](#) - [211](#)

pip, [210](#) – [211](#)

instalando Django, [374](#) - [376](#)

installing Django, [374](#) – [376](#)

instalando Matplotlib, [302](#)

installing Matplotlib, [302](#)

instalando Plotly, [320](#)

installing Plotly, [320](#)

instalando Pygame, [228](#)

installing Pygame, [228](#)

instalando pytest, [211](#)

installing pytest, [211](#)

instalando Solicitudes, [357](#)

installing Requests, [357](#)

Linux, instalar pip, [465 - 466](#)

Linux, installing pip, [465 - 466](#)

actualización, [210](#)

updating, [210](#)

pi_string.py, [187 - 189](#)

pi_string.py , [187](#) – [189](#)

pizza.py , [146](#) - [148](#)

pizza.py , [146](#) – [148](#)

Plataforma.sh. *See* Django: despliegue

Platform.sh. *See* Django: deployment

players.py , [61](#) - [62](#)

players.py , [61](#) – [62](#)

Plotly, [302](#) , [319](#) . *See also* mapear terremotos; rodar dados

Plotly, [302](#) , [319](#) . *See also* mapping earthquakes; rolling dice

tipos de gráficos, [322](#)

chart types, [322](#)

personalización de gráficos, [323](#) , [325](#) - [326](#) , [364](#)

customizing plots, [323](#) , [325](#) – [326](#) , [364](#)

documentación, [368](#)

documentation, [368](#)

fig.show() método, [322](#)

fig.show() method, [322](#)

fig.write_html() método, [327](#)

fig.write_html() method, [327](#)

formato de gráficos

formatting plots

etiquetas de los ejes, [323](#)

axis labels, [323](#)

escalas de colores, [349 - 350](#)

color scales, [349 - 350](#)

hover text, [350 - 351](#) , [365 - 366](#)

hover text, [350 – 351](#) , [365 – 366](#)

enlaces en gráficos, [366 - 367](#)

links in charts, [366 – 367](#)

colores del marcador, [349 - 350](#) , [367](#)

marker colors, [349 – 350](#) , [367](#)

tildes, [325 - 326](#)

tick marks, [325 – 326](#)

títulos, [323](#)

titles, [323](#)

tooltips, [365 - 366](#)

tooltips, [365 – 366](#)

update_layout() método, [325 - 326](#) , [364](#)

update_layout() method, [325 – 326](#) , [364](#)

`update_traces()` método, [367](#)

`update_traces()` method, [367](#)

galería, [320](#)

gallery, [320](#)

histogramas, [322](#)

histograms, [322](#)

instalar, [320](#)

installing, [320](#)

`plotly.express` módulo, [322](#) , [347](#) , [368](#)

`plotly.express` module, [322](#) , [347](#) , [368](#)

`px` alias, [322](#)

`px` alias, [322](#)

`px.bar()` función, [322 - 323](#) , [363 - 367](#)

`px.bar()` function, [322 – 323](#) , [363 – 367](#)

ahorrando cifras, [327](#)

saving figures, [327](#)

`scatter_geo()` función, [347 - 352](#)

`scatter_geo()` function, [347 – 352](#)

`pop()` método, [39 - 40](#)

`pop()` method, [39 – 40](#)

argumentos posicionales, [131 - 133](#) . *See also* funciones: argumentos

positional arguments, [131 – 133](#) . *See also* functions: arguments

Peticiones POST, [406](#) . *See also* Django: formularios

POST requests, [406](#) . *See also* Django: forms

printing_models.py , [143 - 145](#)

printing_models.py , [143 – 145](#)

Proyecto Gutenberg, [196 - 197](#)

Project Gutenberg, [196 – 197](#)

indicaciones, [114 - 115](#)

prompts, [114 – 115](#)

.py extensión de archivo, [15 - 16](#)

.py file extension, [15 – 16](#)

Pygame. *See also Alien Invasion*

Pygame. *See also Alien Invasion*

colores de fondo, [231 - 232](#)

background colors, [231 – 232](#)

clock.tick() método, [230 - 231](#)

clock.tick() method, [230 – 231](#)

colisiones, [266 - 267](#) , [270 - 271](#) , [289 - 290](#)

collisions, [266 – 267](#) , [270 – 271](#) , [289 – 290](#)

creando una ventana vacía, [229 - 230](#)

creating an empty window, [229 – 230](#)

cursor, ocultar, [282 - 283](#)

cursor, hiding, [282 – 283](#)

visualización de texto, [278 - 280](#)

displaying text, [278 – 280](#)

fin de los juegos, [274 - 275](#)

ending games, [274 – 275](#)

bucles de eventos, [229 - 230](#)

event loops, [229 – 230](#)

velocidad de fotogramas, [230 - 231](#)

frame rates, [230 – 231](#)

modo pantalla completa, [245](#)

fullscreen mode, [245](#)

grupos

groups

añadiendo elementos, [249](#) - [250](#)

adding elements, [249](#) – [250](#)

definiendo, [248](#) - [249](#)

defining, [248](#) – [249](#)

dibujar todos los elementos en, [249](#) - [250](#) , [257](#) - [258](#)

drawing all elements in, [249](#) – [250](#) , [257](#) – [258](#)

vaciado, [268](#) - [269](#)

emptying, [268](#) – [269](#)

bucle, [249](#) - [251](#)

looping through, [249 – 251](#)

eliminar elementos de, [250 - 251](#)

removing elements from, [250 – 251](#)

actualizando todos los elementos en, [248 - 249](#)

updating all elements in, [248 – 249](#)

imágenes, [234 - 236](#)

images, [234 – 236](#)

instalar, [228](#)

installing, [228](#)

niveles, [283 - 285](#)

levels, [283 – 285](#)

Botón de reproducción, [278 - 283](#)

Play button, [278 – 283](#)

`print()` llama, [251](#)

`print()` calls in, [251](#)

dejarlo, [244 - 245](#)

quitting, [244 - 245](#)

`rect` objetos, [234 - 235](#)

`rect` objects, [234 - 235](#)

creando desde cero, [247 - 248](#)

creating from scratch, [247 - 248](#)

`get_rect()` método, [234 - 235](#)

`get_rect()` method, [234 - 235](#)

posicionamiento, [234 - 235](#) , [238 - 243](#) , [247 - 248](#) , [256 - 262](#) , [278](#) , [286 - 298](#)

positioning, [234 - 235](#) , [238 - 243](#) , [247 - 248](#) , [256 - 262](#) , [278](#) , [286 - 298](#)

size atributo, [261](#)

size attribute, [261](#)

respondiendo a la entrada, [230](#)

responding to input, [230](#)

eventos, [230](#)

events, [230](#)

pulsaciones de teclas, [238 - 242](#)

keypresses, [238 - 242](#)

clics del ratón, [281 - 283](#)

mouse clicks, [281 - 283](#)

coordenadas de pantalla, [235](#)

screen coordinates, [235](#)

superficies, [230](#)

surfaces, [230](#)

juegos de pruebas, [268](#)

testing games, [268](#)

pytest. *See* código de pruebas

pytest. *See* testing code

Python

Python

>>> prompt, [4](#)

>>> prompt, [4](#)

funciones incorporadas, [467](#)

built-in functions, [467](#)

comprobando la versión instalada, [466](#)

checking installed version, [466](#)

instalando

installing

en Linux, [465 - 466](#)

on Linux, [465 – 466](#)

en macOS, [7 - 11](#) , [464 - 465](#)

on macOS, [7 – 11](#) , [464 – 465](#)

en Windows, [5 - 6](#) , [463 - 464](#)

on Windows, [5 – 6](#) , [463 – 464](#)

intérprete, [15 - 16](#)

interpreter, [15 – 16](#)

palabras clave, [466](#)

keywords, [466](#)

Propuesta de mejora de Python (PEP), [68](#)

Python Enhancement Proposal (PEP), [68](#)

biblioteca estándar, [179 - 180](#)

standard library, [179 – 180](#)

sesiones de terminal, [4](#)

terminal sessions, [4](#)

en Linux, [9](#)

on Linux, [9](#)

en macOS, [7 - 8](#)

on macOS, [7 – 8](#)

en Windows, [6](#)

on Windows, [6](#)

versiones, [4](#)

versions, [4](#)

por qué usar Python, [xxxvi](#)

why use Python, [xxxvi](#)

python_repos.py , [357 - 362](#)

python_repos.py , [357 – 362](#)

python_repos_visual.py , [362 - 367](#)

python_repos_visual.py , [362 – 367](#)

Q

Q

abandonar valores, [118](#)

quit values, [118](#)

R

R

random_walk.py , [312 - 313](#)

random_walk.py , [312 – 313](#)

paseos aleatorios, [312 - 318](#)

random walks, [312 - 318](#)

choice() función, [313](#)

choice() function, [313](#)

puntos de coloración, [315 - 316](#)

coloring points, [315 - 316](#)

fill_walk() método, [312 - 313](#)

fill_walk() method, [312 - 313](#)

generando paseos múltiples, [314 - 315](#)

generating multiple walks, [314 - 315](#)

trazado, [313 - 314](#)

plotting, [313 - 314](#)

RandomWalk clase, [312 - 313](#)

RandomWalk class, [312 – 313](#)

puntos de partida y de llegada, [316 - 317](#)

starting and ending points, [316 – 317](#)

range() función, [58 - 59](#)

range() function, [58 – 59](#)

read_text() método, [185](#) , [195 - 196](#)

read_text() method, [185](#) , [195 – 196](#)

refactorización, [204 - 206](#) , [237 - 238](#) , [260](#) , [269 - 270](#)

refactoring, [204 – 206](#) , [237 – 238](#) , [260](#) , [269 – 270](#)

remember_me.py , [202 - 206](#)

remember_me.py , [202 – 206](#)

removeprefix() método, [24](#)

removeprefix() method, [24](#)

`removesuffix()` método, [25](#)

`removesuffix()` method, [25](#)

Solicita el paquete, instalando, [357](#)

Requests package, installing, [357](#)

valores de retorno, [137](#) - [141](#)

return values, [137](#) – [141](#)

rollercoaster.py , [116](#)

rollercoaster.py , [116](#)

tirando dados, [319](#) - [327](#) . *See also* Plotly

rolling dice, [319](#) – [327](#) . *See also* Plotly

analizando resultados, [321](#) - [322](#)

analyzing results, [321](#) – [322](#)

Die clase, [320](#)

Die class, [320](#)

dados de diferentes tamaños, [326 - 327](#)

different-size dice, [326 – 327](#)

randint() función, [320](#)

randint() function, [320](#)

tirando dos dados, [324 - 326](#)

rolling two dice, [324 – 326](#)

depuración con patito de goma, [478](#)

rubber duck debugging, [478](#)

rstrip() método, [22 - 23](#)

rstrip() method, [22 – 23](#)

rw_visual.py , [313 - 318](#)

rw_visual.py , [313 – 318](#)

S

S

scatter_squares.py , [306 - 311](#)

scatter_squares.py , [306 – 311](#)

conjuntos, [103 - 104](#)

sets, [103 – 104](#)

sitka_highs_lows.py , [336 - 338](#)

sitka_highs_lows.py , [336 – 338](#)

sitka_highs.py , [330 - 336](#)

sitka_highs.py , [330 – 336](#)

sleep() función, [272](#)

sleep() function, [272](#)

rodajas, [61 - 64](#)

slices, [61 – 64](#)

sorted() función, [43 - 44](#), [102 - 103](#)

sorted() function, [43 – 44](#), [102 – 103](#)

sort() método, [43](#)

sort() method, [43](#)

splitlines() método, [186 - 187](#)

splitlines() method, [186 – 187](#)

split() método, [196 - 197](#)

split() method, [196 – 197](#)

Base de datos SQLite, [376 - 377](#)

SQLite database, [376 – 377](#)

square_numbers.py, [58 - 59](#)

square_numbers.py, [58 – 59](#)

`squares.py` , [59 - 60](#)

`squares.py` , [59 - 60](#)

Desbordamiento de pila, [479](#)

Stack Overflow, [479](#)

almacenamiento de datos, [201 - 204](#) . *See also* Archivos JSON

storing data, [201 - 204](#) . *See also* JSON files

guardar y leer datos, [202 - 204](#)

saving and reading data, [202 - 204](#)

cadenas, [19 - 25](#)

strings, [19 - 25](#)

cambiando de caso, [20](#)

changing case, [20](#)

cadenas-f, [20 - 21](#) , [291 - 292](#)

f-strings, [20 – 21](#), [291 – 292](#)

métodos

methods

lower() , [20](#)

lower() , [20](#)

lstrip() , [22 - 23](#)

lstrip() , [22 – 23](#)

removeprefix() , [23 - 24](#)

removeprefix() , [23 – 24](#)

removesuffix() , [25](#)

removesuffix() , [25](#)

rstrip() , [22 - 23](#)

rstrip() , [22 – 23](#)

`split()` , [196 - 197](#)

`split()` , [196 – 197](#)

`splitlines()` , [186 - 187](#)

`splitlines()` , [186 – 187](#)

`strip()` , [22 - 23](#)

`strip()` , [22 – 23](#)

`title()` , [20](#)

`title()` , [20](#)

`upper()` , [20](#)

`upper()` , [20](#)

multilínea, [115](#)

multiline, [115](#)

nuevas líneas dentro, [21 - 22](#)

newlines in, [21](#) – [22](#)

comillas simples y dobles, [19](#) , [24](#) - [25](#)

single and double quotes, [19](#) , [24](#) – [25](#)

pestañas en, [21](#) - [22](#)

tabs in, [21](#) – [22](#)

variables en, [20](#) - [21](#)

variables in, [20](#) – [21](#)

espacio en blanco dentro, [21](#) - [23](#)

whitespace in, [21](#) – [23](#)

strip() método, [22](#) - [23](#)

strip() method, [22](#) – [23](#)

strptime() método, [333](#) - [335](#)

strptime() method, [333](#) – [335](#)

normas de estilo, [68 - 69](#)

style guidelines, [68 – 69](#)

líneas en blanco, [69](#)

blank lines, [69](#)

CamelCase, [181](#)

CamelCase, [181](#)

clases, [181](#)

classes, [181](#)

diccionarios, [96 - 97](#)

dictionaries, [96 – 97](#)

funciones, [153](#)

functions, [153](#)

if declaraciones, [89](#)

if statements, [89](#)

sangría, [68](#)

indentation, [68](#)

longitud de línea, [69](#)

line length, [69](#)

PEP 8, [68](#)

PEP 8, [68](#)

survey.py , [218](#)

survey.py , [218](#)

errores de sintaxis, [24](#)

syntax errors, [24](#)

evitando con cadenas, [24](#) - [25](#)

avoiding with strings, [24](#) – [25](#)

resaltado de sintaxis, [16](#)

syntax highlighting, [16](#)

T

T

`tab ( \t )`, [21 - 22](#)

`tab ( \t )`, [21 - 22](#)

plantillas. *See under* Django

templates. *See under* Django

comprobación de código, [209 - 223](#)

testing code, [209 - 223](#)

aserciones, [213](#) , [217 - 218](#)

assertions, [213](#) , [217 - 218](#)

pruebas fallidas, [214 - 216](#)

failing tests, [214 – 216](#)

cobertura completa, [212](#)

full coverage, [212](#)

pruebas de nomenclatura, [213](#)

naming tests, [213](#)

superando pruebas, [212 - 214](#)

passing tests, [212 – 214](#)

pytest, [209 - 223](#)

pytest, [209 – 223](#)

fixtures, [221 - 223](#)

fixtures, [221 – 223](#)

instalación, [210 - 211](#)

installing, [210 – 211](#)

ejecución de pruebas, [213 - 214](#)

running tests, [213 – 214](#)

casos de prueba, [212](#)

test cases, [212](#)

clases de pruebas, [217 - 223](#)

testing classes, [217 – 223](#)

comprobación de funciones, [211 - 217](#)

testing functions, [211 – 217](#)

pruebas unitarias, [212](#)

unit tests, [212](#)

test_name_function.py, [212 - 217](#)

test_name_function.py, [212 – 217](#)

test_survey.py, [220 - 223](#)

`test_survey.py` , [220](#) – [223](#)

editores de texto e IDEs. *See also* Código VS

text editors and IDEs. *See also* VS Code

Emacs y Vim, [475](#)

Emacs and Vim, [475](#)

Geany, [474](#)

Geany, [474](#)

IDLE, [474](#)

IDLE, [474](#)

Cuadernos Jupyter, [475](#)

Jupyter Notebooks, [475](#)

PyCharm, [475](#)

PyCharm, [475](#)

Texto Sublime, [474](#)

Sublime Text, [474](#)

paquete de terceros, [210](#)

third-party package, [210](#)

toppings.py, [74](#), [82](#) - [83](#)

toppings.py, [74](#), [82](#) - [83](#)

tracebacks, [10](#), [17](#) - [18](#), [192](#), [195](#) - [196](#)

tracebacks, [10](#), [17](#) - [18](#), [192](#), [195](#) - [196](#)

try-except bloques. *See* excepciones

try-except blocks. *See* exceptions

tuplas, [65](#) - [67](#)

tuples, [65](#) - [67](#)

definir, [65](#)

defining, [65](#)

for bucle, [66](#) - [67](#)

for loop, [66](#) – [67](#)

escribiendo encima, [67](#)

writing over, [67](#)

errores tipográficos, [66](#)

type errors, [66](#)

U

U

guión bajo (_)

underscore (_)

en nombres de archivos y carpetas, [10](#)

in file and folder names, [10](#)

en números, [28](#)

in numbers, [28](#)

en nombres de variables, [17](#)

in variable names, [17](#)

pruebas unitarias, [212](#)

unit tests, [212](#)

user_profile.py , [148 - 149](#)

user_profile.py , [148 - 149](#)

V

V

values() método, [103 - 104](#)

values() method, [103 - 104](#)

variables, [16 - 19](#) , [28](#)

variables, [16 – 19](#), [28](#)

constantes, [28](#)

constants, [28](#)

como etiquetas, [18 - 19](#)

as labels, [18 – 19](#)

asignación múltiple, [28](#)

multiple assignment, [28](#)

errores de nombre, [17 - 18](#)

name errors, [17 – 18](#)

convenciones de nomenclatura, [17](#)

naming conventions, [17](#)

valores, [16](#)

values, [16](#)

venv módulo, [374 - 375](#)

venv module, [374 – 375](#)

control de versiones. *See* Git

version control. *See* Git

entornos virtuales, [374 - 375](#)

virtual environments, [374 – 375](#)

voting.py , [78 - 80](#)

voting.py , [78 – 80](#)

VS Code, [4 - 5](#)

VS Code, [4 – 5](#)

configuración, [470 - 473](#)

configuring, [470 – 473](#)

características, [469 - 470](#)

features, [469 – 470](#)

instalando

installing

en Linux, [9](#)

on Linux, [9](#)

en macOS, [8](#)

on macOS, [8](#)

en Windows, [6](#)

on Windows, [6](#)

Extensión de Python, [9 - 10](#)

Python extension, [9 – 10](#)

Abrir archivos con Python, [185](#)

opening files with Python, [185](#)

Extensión de Python, [9](#)

Python extension, [9](#)

archivos en ejecución, [10](#)

running files, [10](#)

atajos, [473](#) - [474](#)

shortcuts, [473](#) – [474](#)

tabuladores y espacios, [471](#)

tabs and spaces, [471](#)

W

W

datos meteorológicos, [330](#) - [341](#) . *See also* Archivos CSV; Matplotlib

weather data, [330](#) – [341](#) . *See also* CSV files; Matplotlib

while bucles, [117](#) - [126](#)

while loops, [117 – 126](#)

active bandera, [120 - 121](#)

active flag, [120 – 121](#)

break declaración, [121](#)

break statement, [121](#)

continue declaración, [122](#)

continue statement, [122](#)

bucles infinitos, [122 - 123](#)

infinite loops, [122 – 123](#)

moviendo elementos entre listas, [124](#)

moving items between lists, [124](#)

abandonar valores, [118](#)

quit values, [118](#)

eliminar todos los elementos de la lista, [125](#)

removing all items from list, [125](#)

espacios en blanco, [21 - 23](#) . *See also* cadenas

whitespace, [21 - 23](#) . *See also* strings

Windows

Windows

rutas de archivos, [186](#)

file paths, [186](#)

Python

Python

configuración, [5 - 6](#) , [9 - 12](#) , [463 - 464](#)

setting up, [5 - 6](#) , [9 - 12](#) , [463 - 464](#)

solución de problemas de instalación, [10](#)

troubleshooting installation, [10](#)

terminales

terminals

ejecutando programas desde, [12](#)

running programs from, [12](#)

iniciar sesión en Python, [6](#)

starting Python session, [6](#)

VS Code, instalando, [6](#)

VS Code, installing, [6](#)

word_count.py, [197 - 199](#)

word_count.py, [197 - 199](#)

write_message.py, [190 - 191](#)

write_message.py, [190 - 191](#)

`write_text()` método, [190 - 191](#)

`write_text()` method, [190 – 191](#)

Z

Z

Zen de Python, [30 - 31](#)

Zen of Python, [30 – 31](#)

`ZeroDivisionError` , [192 - 195](#)

`ZeroDivisionError` , [192 – 195](#)
