Programación en Javascript II

En este manual explicamos todos los recursos con los que cuenta un programador de Javascript para crear todo tipo de efectos y aplicaciones.

Autores del manual

Este manual ha sido realizado por los siguientes colaboradores de DesarrolloWeb.com:

Miguel Angel Alvarez
Director de DesarrolloWeb.com y EscuelaIT
http://www.desarrolloweb.com
(44 capítulos)

Dairo Galeano
Desarrollador independiente
(2 capítulos)

Parte 1:
Introducción a la segunda parte del Manual de Javascript

En esta segunda parte partimos de la base que las personas conocen el lenguaje y la sintaxis y vamos a ver cómo utilizarlo para hacer programación de páginas enriquecidas del lado del cliente.

1.1.- Introducción al manual II de Javascript

Empezamos el segundo manual de Javascrip con un repaso a los temas que vamos a cubrir.

En esta segunda parte del manual de Javascript vamos a tratar de explicar todos los recursos con los que cuenta un programador de Javascript y con los que puede crear todo tipo de efectos y aplicaciones.

Para leer y entender bien lo que viene en los siguientes capítulos es necesario haber leído antes la primera parte de este manual: Programación en Javascript I, donde se explican las bases sobre las que tenemos que asentar los siguientes conocimientos. En la primera parte de este manual conocimos los orígenes y las aplicaciones de Javascript, pero sobretodo hicimos hincapié en su sintaxis, muy importante para entender los scripts que haremos en los siguientes capítulos.

Los objetivos de los siguientes capítulos cubrirán aspectos diversos de Javascript como:

- Funciones incorporadas en el lenguaje Javascript
- Los objetos en Javascript
- Jerarquía de objetos del navegador
- Trabajo con formularios
- Control de ventanas secundarias y frames
- Eventos

Como se puede ver, todos los temas tienen un fuerte carácter práctico y cubren aspectos varios con los que formarnos a nivel avanzado en Javascript. Esperamos que sirvan para iluminar un área tan amplia del desarrollo de páginas web como es el scripting del lado del cliente.

Además, ya fuera de este manual, te recordamos que en DesarrolloWeb.com tienes disponibles muchos más artículos y manuales dedicados a este lenguaje que te servirán para profundizar en la materia, como los frameworks jQuery o Mootools, trabajo con imágenes, canvas, así como videotutoriales para los que prefieran aprender con vídeo. A todo ello podrás acceder desde la sección Javascript a fondo!

Vamos sin más pausa con esta segunda parte del manual, que resultará mucho más entretenida y práctica que la primera.

Artículo por Miguel Angel Alvarez

Parte 2:
Clases y objetos nativos de Javascript

Javascript dispone de un conjunto de clases y objetos que podemos utilizar para realizar operaciones diversas. Todos ellos forman una librería de código que nos facilitará la vida cuando hagamos los programas o scripts.

2.1.- Libreria de funciones Javascript

Javascript, al igual que cualquier otro lenguaje, pone a nuestra disposición un conjunto de funciones que llamamos funciones nativas de Javascript.

En todos los lenguajes de programación existen librerías de funciones que sirven para hacer cosas diversas y muy repetitivas a la hora de programar. Las librerías de los lenguajes de programación ahorran la tarea de escribir las funciones comunes que por lo general pueden necesitar los programadores. Un lenguaje de programación bien desarrollado tendrá una buena cantidad de ellas. En ocasiones es más complicado conocer bien todas las librerías que aprender a programar en el lenguaje.

Javascript contiene una buena cantidad de funciones en sus librerías. Como se trata de un lenguaje que trabaja con objetos muchas de las librerías se implementan a través de objetos. Por ejemplo, las funciones matemáticas o las de manejo de strings se implementan mediante los objetos Math y String. Sin embargo, existen algunas funciones que no están asociadas a ningún objeto y son las que veremos en este capítulo, ya que todavía no conocemos los objetos y no los necesitaremos para estudiarlas.

Estas son las funciones que Javascript pone a disposición de los programadores.

eval(string)
Esta función recibe una cadena de caracteres y la ejecuta como si fuera una sentencia de Javascript.

parseInt(cadena,base)
Recibe una cadena y una base. Devuelve un valor numérico resultante de convertir la cadena en un número en la base indicada.

parseFloat(cadena)
Convierte la cadena en un número y lo devuelve.

escape(carácter)
Devuelve un el carácter que recibe por parámetro en una codificación ISO Latin 1.

unescape(carácter)
Hace exatamente lo opuesto a la función escape.

isNaN(número)
Devuelve un boleano dependiendo de lo que recibe por parámetro. Si no es un número devuelve un true, si es un numero devuelve false.

Las librerías que se implementan mediante objetos y las del manejo del explorador, que también se manejan con objetos, las veremos más adelante.

> **Nota:** No queremos llevar a engaño a las personas con esta corta lista de funciones nativas de Javascript. Realmente existen muchas otras funciones que vamos a ver a lo largo del presente manual, lo que ocurre es que están asociadas a objetos. Por ejemplo, como habíamos señalado, existen funciones de cadenas de caracteres, que están asociadas a objetos string, funciones para trabajo con cálculos matemáticos avanzados, que están asociadas a la clase Math, funciones para trabajo con el objeto de la ventana del navegador, con el documento, etc.

A continuación vamos a ver algún ejemplo con las funciones más importantes de esta lista.

Artículo por Miguel Angel Alvarez

2.2.- Ejemplos de funciones de la librería Javascript

Las funciones nativas de Javascript son bastante importantes. Ahora Veremos varios ejemplos de funciones de la librería: eval(), parseInt(), isNaN().

En el anterior artículo del Manual de programación en Javascript II vimos un listado de las funciones nativas del lenguaje Javascript. Ahora podemos ver varios ejemplos de utilización de funciones nativas de Javascript, que tenemos disponibles en cualquier navegador y en cualquier versión de Javascript.

Veremos tres funciones de diverso ámbito que resultan bastante fundamentales en el trabajo habitual con este lenguaje, explicadas a través de ejemplos.

2.2.1.- Función eval

Esta función es muy importante, tanto que hay algunas aplicaciones de Javascript que no se podrían realizar si no la utilizamos. Su utilización es muy simple, pero puede que resulte un poco más complejo entender en qué casos utilizarla porque a veces resulta un poco sutil su aplicación.

Con los conocimientos actuales no podemos hacer un ejemplo muy complicado, pero por lo menos podemos ver en marcha la función. Vamos a utilizarla en una sentencia un poco rara y bastante inservible, pero si la conseguimos entender conseguiremos entender también la función eval.

```
var miTexto = "3 + 5"
eval("document.write(" + miTexto +")")
```

Primero creamos una variable con un texto, en la siguiente línea utilizamos la función eval y como parámetro le pasamos una instrucción javascript para escribir en pantalla. Si concatenamos los strings que hay dentro de los paréntesis de la función eval nos queda esto.

```
document.write(3 + 5)
```

La función eval ejecuta la instrucción que se le pasa por parámetro, así que ejecutará esta sentencia, lo que dará como resultado que se escriba un 8 en la página web. Primero se resuelve la suma que hay entre paréntesis, con lo que obtenemos el 8 y luego se ejecuta la instrucción de escribir en pantalla.

2.2.2.- Función parseInt

Esta función recibe un número, escrito como una cadena de caracteres, y un número que indica una base. En realidad puede recibir otros tipos de variables, dado que las variables no tienen tipo en Javascript, pero se suele utilizar pasándole un string para convertir la variable de texto en un número.

Las distintas bases que puede recibir la función son 2, 8, 10 y 16. Si no le pasamos ningún valor como base la función interpreta que la base es decimal. El valor que devuelve la función siempre tiene base 10, de modo que si la base no es 10 convierte el número a esa base antes de devolverlo.

Veamos una serie de llamadas a la función parseInt para ver lo que devuelve y entender un poco más la función.

```
document.write (parseInt("34"))
```
Devuelve el numero 34

```
document.write (parseInt("101011",2))
```
Devuelve el numero 43

```
document.write (parseInt("34",8))
```
Devuelve el numero 28

```
document.write (parseInt("3F",16))
```
Devuelve el numero 63

Esta función se utiliza en la práctica para un montón de cosas distintas en el manejo con números, por ejemplo obtener la parte entera de un decimal.

```
document.write (parseInt("3.38"))
```
Devuelve el numero 3

También es muy habitual su uso para saber si una variable es numérica, pues si le pasamos un texto a la función que no sea numérico nos devolverá NaN (Not a Number) lo que quiere decir que No es un Número.

```
document.write (parseInt("desarrolloweb.com"))
```
Devuelve el numero NaN

Este mismo ejemplo es interesante con una modificación, pues si le pasamos una combinación de letras y números nos dará lo siguiente.

```
document.write (parseInt("16XX3U"))
```
Devuelve el numero 16

document.write (parseInt("TG45"))
Devuelve el numero NaN

Como se puede ver, la función intenta convertir el string en número y si no puede devuelve NaN.

Todos estos ejemplos, un tanto inconexos, sobre cómo trabaja parseInt los revisaremos más adelante en ejemplos más prácticos cuando tratemos el trabajo con formularios.

2.2.3.- Función isNaN

Esta función devuelve un boleano dependiendo de si lo que recibe es un número o no. Lo único que puede recibir es un número o la expresión NaN. Si recibe un NaN devuelve true y si recibe un número devuelve false. Es una función muy sencilla de entender y de utilizar.

La función suele trabajar en combinación con la función parseInt o parseFloat, para saber si lo que devuelven estas dos funciones es un número o no.

```
miInteger = parseInt("A3.6")
isNaN(miInteger)
```

En la primera línea asignamos a la variable miInteger el resultado de intentar convertir a entero el texto A3.6. Como este texto no se puede convertir a número la función parseInt devuelve NaN. La segunda línea comprueba si la variable anterior es NaN y como si que lo es devuelve un true.

```
miFloat = parseFloat("4.7")
isNaN(miFloat)
```

En este ejemplo convertimos un texto a número con decimales. El texto se convierte perfectamente porque corresponde con un número. Al recibir un número la función isNaN devuelve un false.

Referencia: Validar entero en campo de formulario
Tenemos un Taller de Javascript muy interesante que ha sido realizado para afianzar los conocimientos de estos capítulos. Se trata de un script para validar un campo de formulario de manera que sepamos seguro que dentro del campo hay siempre un número entero. Puede ser muy interesante leerlo ahora, ya que utilizamos las funciones isNaN() y parseInt(). Ver el taller

Esperamos que los ejemplos vistos en este artículo hayan resultado interesantes. No obstante, como habíamos señalado anteriormente, existen bastantes otras funciones nativas en Javascript que debemos conocer, pero que están asociadas a clases y objetos nativos Javascript. Pero antes de pasar a ese punto queremos ofrecer una pequeña guía básica para el trabajo con programación orientada a objetos en Javascript.

Artículo por Miguel Angel Alvarez

2.3.- Objetos en Javascript

Vemos una primera introducción al mundo de los objetos en general y en particular en el lenguaje Javascript.

Vamos a introducirnos en un tema muy importante de Javascript como son los objetos. Es un tema que aun no hemos visto y sobre el que en adelante vamos a tratar constantemente pues todas las cosas en Javascript, incluso las más sencillas, las vamos a realizar a través del manejo de objetos. De hecho, en los ejemplos realizados hasta ahora hemos hecho grandes esfuerzos para no utilizar objetos y aun así los hemos utilizado en alguna ocasión, pues es muy difícil encontrar ejemplos en Javascript que, aunque sean simples, no hagan uso de ellos.

La programación orientada a objetos (POO) representa una nueva manera de pensar a la hora de hacer un programa. Javascript no es un lenguaje de programación orientado a objetos, aunque los utiliza en muchas ocasiones: podemos

crear nuevos objetos y utilizar muchos que están creados desde un principio. Sin embargo la manera de programar no va a cambiar mucho y lo que hemos visto hasta aquí relativo a sintaxis, funciones, etc. puede ser utilizado igual que se ha indicado. Solo vamos a aprender una especie de estructura nueva.

Para empezar a empaparnos en este asunto es imprescindible que nos leamos un pequeño artículo publicado en DesarrolloWeb sobre la programación orientada a objetos. Después de su lectura puedes continuar con estas líneas y si conoces ya la POO continúa leyendo sin pausa.

2.3.1.- Cómo instanciar objetos

Instanciar un objeto es la acción de crear un ejemplar de una clase para poder trabajar con él luego. Recordamos que un objeto se crea a partir de una clase y la clase es la definición de las características y funcionalidades de un objeto. Con las clases no se trabaja, estas sólo son definiciones, para trabajar con una clase debemos tener un objeto instanciado de esa clase.

En javascript para crear un objeto a partir de una clase se utiliza la instrucción new, de esta manera.

```
var miObjeto = new miClase()
```

En una variable que llamamos miObjeto asigno un nuevo (new) ejemplar de la clase miClase. Los paréntesis se rellenan con los datos que necesite la clase para inicializar el objeto, si no hay que meter ningún parámetro los paréntesis se colocan vacíos. En realidad lo que se hace cuando se crea un objeto es llamar al constructor de esa clase y el constructor es el encargado de crearlo e inicializarlo. Hablaremos sobre esto más adelante.

2.3.2.- Cómo acceder a propiedades y métodos de los objetos

En Javascript podemos acceder a las propiedades y métodos de objetos de forma similar a como se hace en otros lenguajes de programación, con el operador punto (".").

Las propiedades se acceden colocando el nombre del objeto seguido de un punto y el nombre de la propiedad que se desea acceder. De esta manera:

```
miObjeto.miPropiedad
```

Para llamar a los métodos utilizamos una sintaxis similar pero poniendo al final entre paréntesis los parámetros que pasamos a los métodos. Del siguiente modo:

```
miObjeto.miMetodo(parametro1,parametro2)
```

Si el método no recibe parámetros colocamos los paréntesis también, pero sin nada dentro.

```
miObjeto.miMetodo()
```

> **Nota:** Obviamente, el mundo de la programación orientada a objetos en Javascript es mucho más amplio de lo que hemos visto en este artículo. Al menos con lo que hemos aprendido tenemos suficiente para encarar los artículos siguientes del Manual de programación Javascript avanzada.

En el siguiente artículo veremos una introducción objetos nativos en Javascript y luego empezaremos a verlos uno por uno, de tal modo que podremos practicar todo lo que hemos visto hasta ahora sobre POO.

Artículo por Miguel Angel Alvarez

2.4.- Objetos incorporados en Javascript

Lista de los objetos que tenemos a nuestra disposición a la hora de trabajar con Javascript.

Llegamos a un punto interesante dentro de la segunda parte del Manual de programación en Javascript. En estos momentos sabemos ya lo que es la programación orientada a objetos, así que vamos a introducirnos en el manejo de objetos en Javascript y para ello vamos a empezar con el estudio de las clases predefinidas que implementan las librerías para el trabajo con strings, matemáticas, fechas etc.

Las clases que se encuentran disponibles de manera nativa en Javascript, y que vamos a ver a continuación, son las siguientes:

- **String**, para el trabajo con cadenas de caracteres.
- **Date**, para el trabajo con fechas.
- **Math**, para realizar funciones matemáticas.
- **Number**, para realizar algunas cosas con números
- **Boolean**, trabajo con boleanos.

Nota: Las clases se escriben con la primera letra en mayúsculas. Tiene que quedar claro que una clase es una especie de "declaración de características y funcionalidades" de los objetos. Dicho de otro modo, las clases son descripciones de la forma y funcionamiento de los objetos. Con las clases generalmente no se realiza ningún trabajo, sino que se hace con los objetos, que creamos a partir de las clases.

Una vez comprendida la diferencia entre objetos y clases, cabe señalar que String es una clase, lo mismo que Date. Si queremos trabajar con cadenas de caracteres o fechas necesitamos crear objetos de las clases String y Date respectivamente. Como sabemos, Javascript es un lenguaje sensible a las mayúsculas y las minúsculas y en este caso, **las clases, por convención, siempre se escriben con la primera letra en mayúscula y también la primera letra de las siguientes palabras**, si es que el nombre de la clase está compuesto de varias palabras. Por ejemplo si tuvieramos la clase de "Alumnos universitarios" se escribiría con la -A- de alumnos y la -U- de universitarios en mayúscula. AlumnosUniversitarios sería el nombre de nuestra supuesta clase.

Hay otros que pertenecen a este mismo conjunto, los enumeramos aquí para que quede constancia de ellos, pero no los vamos a tocar en capítulos siguientes.

- **Array**, ya los hemos visto en capítulos correspondientes al primer manual de Javascript.
- También la clase **Function**, lo hemos visto parcialmente al estudiar las funciones.
- Hay otra clase **RegExp** que sirve para construir patrones que veremos tal vez junto con Function cuando tratemos temas más avanzados todavía.

Nota: Uso de mayúsculas y minúsculas. Ya que nos hemos parado anteriormente a hablar sobre el uso de mayúsculas en ciertas letras de los nombre de las clases, vamos a terminar de explicar la convención que se lleva a cabo en Javascript para nombrar a los elementos.

Para empezar, cualquier variable se suele escribir en minúsculas, aunque si este nombre de variable se compone de varias palabras, se suele escribir en mayúscula la primera letra de las siguientes palabras a la primera. Esto se hace así en cualquier tipo de variable o nombre menos en los nombres de las clases, donde se escribe también en mayúscula el primer caracter de la primera palabra.

Ejemplos:

Number, que es una clase se escribe con la primera en mayúscula.
RegExp, que es el nombre de otra clase compuesto por dos palabras, tiene la primera letras de las dos palabras en mayúscula.
miCadena, que supongamos que es un objeto de la clase String, se escribe con la primera letra en minúscula y la primera letra de la segunda palabra en mayúscula.
fecha, que supongamos que es un objeto de la clase Date, se escribe en minúsculas por ser un objeto.
miFunción(), que es una función definida por nosotros, se acostumbra a escribir con minúscula la primera.

Como decimos, esta es la norma general para dar nombres a las variables, clases, objetos, funciones, etc. en Javascript. Si la seguimos así, no tendremos problemas a la hora de saber si un nombre en Javascript tiene o no alguna mayúscula y además todo nuestro trabajo será más claro y fácil de seguir por otros programadores o nosotros mismos en caso de retomar un código una vez pasado un tiempo.

Artículo por Miguel Angel Alvarez

2.5.- Clase String en Javascript

La clase que nos sive para manejar cadenas de caracteres. Estudiamos sus propiedades y la lista completa de métodos.

En javascript las variables de tipo texto son objetos de la clase String. Esto quiere decir que cada una de las variables que creamos de tipo texto tienen una serie de propiedades y métodos. Recordamos que las propiedades son las características, como por ejemplo longitud en caracteres del string y los métodos son funcionalidades, como pueden ser extraer un substring o poner el texto en mayúsculas.

Para crear un objeto de la clase String lo único que hay que hacer es asignar un texto a una variable. El texto va entre comillas, como ya hemos visto en los capítulos de sintaxis. También se puede crear un objeto string con el operador new, que veremos más adelante. La única diferencia es que en versiones de Javascript 1.0 no funcionará new para crear los Strings.

2.5.1.- Propiedades de String

Length
La clase String sólo tiene una propiedad: length, que guarda el número de caracteres del String.

2.5.2.- Métodos de String

Los objetos de la clase String tienen una buena cantidad de métodos para realizar muchas cosas interesantes. Primero vamos a ver una lista de los métodos más interesantes y luego vamos a ver otra lista de métodos menos útiles.

charAt(indice)
Devuelve el carácter que hay en la posición indicada como índice. Las posiciones de un string empiezan en 0.

indexOf(carácter,desde)
Devuelve la posición de la primera vez que aparece el carácter indicado por parámetro en un string. Si no encuentra el carácter en el string devuelve -1. El segundo parámetro es opcional y sirve para indicar a partir de que posición se desea que empiece la búsqueda.

lastIndexOf(carácter,desde)
Busca la posición de un carácter exáctamente igual a como lo hace la función indexOf pero desde el final en lugar del principio. El segundo parámetro indica el número de caracteres desde donde se busca, igual que en indexOf.

replace(substring_a_buscar,nuevoStr)
Implementado en Javascript 1.2, sirve para reemplazar porciones del texto de un string por otro texto, por ejemplo, podríamos uilizarlo para reemplazar todas las apariciones del substring "xxx" por "yyy". El método no reemplaza en el string, sino que devuelve un resultante de hacer ese reemplazo. Acepta expresiones regulares como substring a buscar.

split(separador)
Este método sólo es compatible con javascript 1.1 en adelante. Sirve para crear un vector a partir de un String en el que cada elemento es la parte del String que está separada por el separador indicado por parámetro.

substring(inicio,fin)
Devuelve el substring que empieza en el carácter de inicio y termina en el carácter de fin. Si intercambiamos los parámetros de inicio y fin también funciona. Simplemente nos da el substring que hay entre el carácter menor y el mayor.

toLowerCase()
Pone todas los caracteres de un string en minúsculas.

toUpperCase()
Pone todas los caracteres de un string en mayúsculas.

toString()
Este método lo tienen todos los objetos y se usa para convertirlos en cadenas.

Hasta aquí hemos visto los métodos que nos ayudarán a tratar cadenas. Ahora vamos a ver otros métodos que son menos útiles, pero hay que indicarlos para que quede constancia de ellos. Todos sirven para aplicar estilos a un texto y es como si utilizásemos etiquetas HTML. Veamos cómo.

anchor(name)
Convierte en un ancla (sitio a donde dirigir un enlace) una cadena de caracteres usando como el atributo name de la etiqueta <A> lo que recibe por parámetro.

big()
Aumenta el tamaño de letra del string. Es como si colocásemos en un string al principio la etiqueta <BIG> y al final </BIG>.

blink()
Para que parpadee el texto del string, es como utilizar la etiqueta <BLINK>. Solo vale para Netscape.

bold()
Como si utilizásemos la etiqueta <B>.

fixed()
Para utilizar una fuente monoespaciada, etiqueta <TT>.

fontColor(color)
Pone la fuente a ese color. Como utilizar la etiqueta <FONT color=el_color_indicado>.

fontSize(tamaño)
Pone la fuente al tamaño indicado. Como si utilizásemos la etiqueta <FONT> con el atributo size.

italics()
Pone la fuente en cursiva. Etiqueta <I>.

link(url)
Pone el texto como un enlace a la URL indicada. Es como si utilizásemos la etiqueta <A> con el atributo href indicado como parámetro.

small()
Es como utilizar la etiqueta <SMALL>

strike()
Como utilizar la etiqueta <STRIKE>, que sirve para que el texto aparezca tachado.

sub()
Actualiza el texto como si se estuviera utilizando la etiqueta <SUB>, de subíndice.

sup()
Como si utilizásemos la etiqueta <SUP>, de superíndice.

Artículo por Miguel Angel Alvarez

2.6.- Ejemplos de funcionamiento de la clase String

Realizamos un par de scripts en Javascript para ilustrar el funcionamiento de algunos métodos y propiedades de la clase String.

En capítulos anteriores de esta segunda parte del Manual de Javascript vimos lo que son las clases nativas de Javascript. Una de ellas era la clase String cuyo funcionamiento vamos a mostrar a continuación.

Así pues, pasemos a ver unos ejemplos sobre cómo se utilizan los métodos y propiedades del objeto String.

2.6.1.- Ejemplo de strings 1

Vamos a escribir el contenido de un string con un carácter separador ("-") entre cada uno de los caracteres del string.

```
var miString = "Hola Amigos"
var result = ""

for (i=0;i<miString.length-1;i++) {
   result += miString.charAt(i)
   result += "-"
}
result += miString.charAt(miString.length - 1)

document.write(result)
```

Primero creamos dos variables, una con el string a recorrer y otra con un string vacío, donde guardaremos el resultado. Luego hacemos un bucle que recorre desde el primer hasta el penúltimo carácter del string -utilzamos la propiedad length para conocer el número de caracteres del string- y en cada iteración colocamos un carácter del string seguido de un carácter separador "-". Como aun nos queda el último carácter por colocar lo ponemos en la siguiente línea después del bucle. Utilizamos la función charAt para acceder a las posiciones del string. Finalmente imprimimos en la página el resultado.

Podemos ver el ejemplo en funcionamiento en una página a parte.

2.6.2.- Ejemplo de strings 2

Vamos a hacer un script que rompa un string en dos mitades y las imprima por pantalla. Las mitades serán iguales, siempre que el string tenga un número de caracteres par. En caso de que el número de caracteres sea impar no se podrá hacer la mitad exacta, pero partiremos el string lo más aproximado a la mitad.

```
var miString = "0123456789"
var mitad1,mitad2

posicion_mitad = miString.length / 2

mitad1 = miString.substring(0,posicion_mitad)
mitad2 = miString.substring(posicion_mitad,miString.length)

document.write(mitad1 + "<br>" + mitad2)
```

Las dos primeras líneas sirven para declarar las variables que vamos a utilizar e inicializar el string a partir. En la siguiente línea hallamos la posición de la mitad del string.

En las dos siguientes líneas es donde realizamos el trabajo de poner en una variable la primera mitad del string y en la otra la segunda. Para ello utilizamos el método substring pasándole como inicio y fin en el primer caso desde 0 hasta la mitad y en el segundo desde la mitad hasta el final. Para finalizar imprimimos las dos mitades con un salto de línea entre ellas.

Podemos ver el ejemplo en funcionamiento en una página a parte.

Una vez sabemos manejar los objetos de la clase string, podemos pasar a ver otras de las clases nativas de Javascript, como la clase Date.

Artículo por Miguel Angel Alvarez

2.7.- Clase Date en Javascript

Explicamos la clase que se utiliza en Javascript para el manejo de fechas y horas, comentando sus métodos y propiedades.

Vamos a empezar a relatar todas las cosas que debes saber sobre otro de los objetos nativos de Javascript, el que implementa la clase Date.

Sobre la clase Date recae todo el trabajo con fechas en Javascript, como obtener una fecha, el día la hora actuales y otras cosas. Para trabajar con fechas necesitamos instanciar un objeto de la clase Date y con él ya podemos realizar las operaciones que necesitemos.

Un objeto de la clase Date se puede crear de dos maneras distintas. Por un lado podemos crear el objeto con el día y hora actuales y por otro podemos crearlo con un día y hora distintos a los actuales. Esto depende de los parámetros que pasemos al construir los objetos.

Para crear un objeto fecha con el día y hora actuales colocamos los paréntesis vacíos al llamar al constructor de la clase Date.

```
miFecha = new Date()
```

Para crear un objeto fecha con un día y hora distintos de los actuales tenemos que indicar entre paréntesis el momento con que inicializar el objeto. Hay varias maneras de expresar un día y hora válidas, por eso podemos construir una fecha guiándonos por varios esquemas. Estos son dos de ellos, suficientes para crear todo tipo de fechas y horas.

```
miFecha = new Date(año,mes,dia,hora,minutos,segundos)
miFecha = new Date(año,mes,dia)
```

Los valores que debe recibir el constructor son siempre numéricos. Un detalle, el mes comienza por 0, es decir, enero es el mes 0. Si no indicamos la hora, el objeto fecha se crea con hora 00:00:00.

Los objetos de la clase Date no tienen propiedades pero si un montón de métodos, vamos a verlos ahora.

getDate()
Devuelve el día del mes.

getDay()
Devuelve el día de la semana.

getHours()
Retorna la hora.

getMinutes()
Devuelve los minutos.

getMonth()
Devuelve el mes (atención al mes que empieza por 0).

getSeconds()
Devuelve los segundos.

getTime()

Devuelve los milisegundos transcurridos entre el día 1 de enero de 1970 y la fecha correspondiente al objeto al que se le pasa el mensaje.

getYear()
Retorna el año, al que se le ha restado 1900. Por ejemplo, para el 1995 retorna 95, para el 2005 retorna 105. Este método está obsoleto en Netscape a partir de la versión 1.3 de Javascript y ahora se utiliza getFullYear().

getFullYear()
Retorna el año con todos los dígitos. Usar este método para estar seguros de que funcionará todo bien en fechas posteriores al año 2000.

setDate()
Actualiza el día del mes.

setHours()
Actualiza la hora.

setMinutes()
Cambia los minutos.

setMonth()
Cambia el mes (atención al mes que empieza por 0).

setSeconds()
Cambia los segundos.

setTime()
Actualiza la fecha completa. Recibe un número de milisegundos desde el 1 de enero de 1970.

setYear()
Cambia el año recibe un número, al que le suma 1900 antes de colocarlo como año de la fecha. Por ejemplo, si recibe 95 colocará el año 1995. Este método está obsoleto a partir de Javascript 1.3 en Netscape. Ahora se utiliza setFullYear(), indicando el año con todos los dígitos.

setFullYear()
Cambia el año de la fecha al número que recibe por parámetro. El número se indica completo ej: 2005 o 1995. Utilizar este método para estar seguros que todo funciona para fechas posteriores a 2000.

Como habréis podido apreciar, hay algún método obsoleto por cuestiones relativas al año 2000: setYear() y getYear(). Estos métodos se comportan bien en Internet Explorer y no nos dará ningún problema el utilizarlos. Sin embrago, no funcionarán correctamente en Netscape, por lo que es interesante que utilicemos en su lugar los métodos getFullYear() y setFullYear(), que funcionan bien en Netscape e Internet Explorer.

> **Referencia:** Te recomendamos un artículo sobre fechas que es especialmente práctico para las personas que quieren mostrar la fecha actual con varios formatos distintos.

A continuación, para afianzar todos estos conocimientos, te recomendamos ver algunos ejemplos de trabajo con objetos de la clase Date.

Artículo por Miguel Angel Alvarez

2.8.- Ejemplo de funcionamiento de Date

Ejercicio realizado para ilustrar el funcionamiento de la clase Date en Javascript.

En el capítulo anterior de la segunda parte del manual de Javascript explicamos las generalidades de la clase Date, para trabajo con fechas. Así que ahora podemos poner los conocimientos en la práctica en un ejemplo completo.

En este ejemplo vamos a crear dos fechas, una con el instante actual y otra con fecha del pasado. Luego las imprimiremos las dos y extraeremos su año para imprimirlo también. Luego actualizaremos el año de una de las fechas y la volveremos a escribir con un formato más legible.

```
//en estas líneas creamos las fechas
miFechaActual = new Date()
miFechaPasada = new Date(1998,4,23)

//en estas líneas imprimimos las fechas.
document.write (miFechaActual)
document.write ("<br>")
document.write (miFechaPasada)

//extraemos el año de las dos fechas
anoActual = miFechaActual.getFullYear()
anoPasado = miFechaPasada.getFullYear()

//Escribimos en año en la página
document.write("<br>El año actual es: " + anoActual)
document.write("<br>El año pasado es: " + anoPasado)

//cambiamos el año en la fecha actual
miFechaActual.setFullYear(2005)

//extraemos el día mes y año
dia = miFechaActual.getDate()
mes = parseInt(miFechaActual.getMonth()) + 1
ano = miFechaActual.getFullYear()

//escribimos la fecha en un formato legible
document.write ("<br>")
document.write (dia + "/" + mes + "/" + ano)
```

Si se desea, se puede ver en funcionamiento este script en una página a parte.

Hay que destacar un detalle antes de acabar y es que el número del mes puede empezar desde 0. Por lo menos en el Netscape con el que realizamos las pruebas empezaba en 0 el mes. Por esta razón sumamos uno al mes que devuelve el método getMonth.

Hay más detalles a destacar, pues resulta que en Netscape el método getFullYear() devuelve los años trascurridos desde 1900, con lo que al obtener el año de una fecha de, por ejemplo, 2005, indica que es el año 105. Para obtener el año completo tenemos a nuestra disposición el método getFullYear() que devolvería 2005 del mismo modo en Netscape e Internet Explorer.

Mucha atención, pues, al trabajo con fechas en distintas plataformas, puesto que podría ser problemático el hecho de que nos ofrezcan distintas salidas los métodos de manejo de fechas, com siempre dependiendo de la marca y versión de nuestro navegador.

Referencia: Se puede ver otro ejemplo de trabajo con la clase Date en el taller Calcular la edad en Javascript

Artículo por Miguel Angel Alvarez

2.9.- Clase Math en Javascript

La clase que utilizamos para realizar cálculos matemáticos de todo tipo.

La clase Math es una de las clases nativas de Javascript. Proporciona los mecanismos para realizar operaciones matemáticas en Javascript. Algunas operaciones se resuelven rápidamente con los operadores aritméticos que ya conocemos, como la multiplicación o la suma, pero hay una serie de operaciones matemáticas adicionales que se tienen que realizar usando la clase Math como pueden ser calcular un seno o hacer una raiz cuadrada.

De modo que para cualquier cálculo matemático complejo utilizaremos la clase Math, con una particularidad. Hasta ahora cada vez que queríamos hacer algo con una clase debíamos instanciar un objeto de esa clase y trabajar con el objeto y en el caso de la clase Math se trabaja directamente con la clase. Esto se permite por que las propiedades y métodos de la clase Math son lo que se llama propiedades y métodos de clase y para utilizarlos se opera a través de la clase en lugar de los objetos. Dicho de otra forma, para trabajar con la clase Math no deberemos utilizar la instrucción new y utilizaremos el nombre de la clase para acceder a sus propiedades y métodos.

2.9.1.- Propiedades de Math

Las propiedades guardan valores que probablemente necesitemos en algún momento si estamos haciendo cálculos matemáticos. Es probable que estas propiedades resulten un poco raras a las personas que desconocen las matemáticas avanzadas, pero los que las conozcan sabrán de su utilidad.

E
Número E o constante de Euler, la base de los logaritmos neperianos.

LN2
Logaritmo neperiano de 2.

LN10
Logaritmo neperiano de 10.

LOG2E
Logaritmo en base 2 de E.

LOG10E
Logaritmo en base 10 de E.

PI
Conocido número para cálculo con círculos.

SQRT1_2
Raiz cuadrada de un medio.

SQRT2
Raiz cuadrada de 2.

2.9.2.- Métodos de Math

Así mismo, tenemos una serie de métodos para realizar operaciones mateméticas típicas, aunque un poco complejas. Todos los que conozcan las matemáticas a un buen nivel conocerán el significado de estas operaciones.

abs()
Devuelve el valor absoluto de un número. El valor después de quitarle el signo.

acos()
Devuelve el arcocoseno de un número en radianes.

asin()
Devuelve el arcoseno de un numero en radianes.

atan()
Devuelve un arcotangente de un numero.

ceil()
Devuelve el entero igual o inmediatamente siguiente de un número. Por ejemplo, ceil(3) vale 3, ceil(3.4) es 4.

cos()
Retorna el coseno de un número.

exp()
Retorna el resultado de elevar el número E por un número.

floor()
Lo contrario de ceil(), pues devuelve un número igual o inmediatamente inferior.

log()
Devuelve el logaritmo neperiano de un número.

max()
Retorna el mayor de 2 números.

min()
Retorna el menor de 2 números.

pow()
Recibe dos números como parámetros y devuelve el primer número elevado al segundo número.

random()
Devuelve un número aleatorio entre 0 y 1. Método creado a partir de Javascript 1.1.

round()
Redondea al entero más próximo.

sin()
Devuelve el seno de un número con un ángulo en radianes.

sqrt()
Retorna la raiz cuadrada de un número.

tan()
Calcula y devuelve la tangente de un número en radianes.

Ejemplo de utilización de la clase Math

Vamos a ver un sencillo ejemplo sobre cómo utilizar métodos y propiedaes de la clase Math para calcular el seno y el coseno de 2 PI radianes (una vuelta completa). Como algunos de vosotros sabréis, el coseno de 2 PI radianes debe dar como resultado 1 y el seno 0.

```
document.write (Math.cos(2 * Math.PI))
```

document.write ("
")

document.write (Math.sin(2 * Math.PI))

2 PI radianes es el resultado de multiplicar 2 por el número PI. Ese resultado es lo que recibe el método cos, y da como resultado 1. En el caso del seno el resultado no es exactamente 0 pero está aproximado con una exactitud de más de una millonésima de fracción. Se escriben los el seno y coseno con un salto de línea en medio para que quede más claro.

Podemos ver el resultado de ejecutar estas líneas de código.

Además, si deseamos profundizar en la clase Math os recomiendo leer el artículo de crear un número aleatorio. También

se hace uso de la clase Math en el artículo enlace aleatorio. Todos en el Taller de Javascript.

Artículo por Miguel Angel Alvarez

2.10.- Clase Number en Javascript

Clase que modeliza el tipo de datos numérico.

Vamos a ver a continuación otra de las clases nativas de Javascript para trabajo con datos numéricos. Se trata de la clase Number, que modeliza el tipo de datos numérico y fue añadida en la versión 1.1 de Javascript (con Netscape Navigator 3).

Como veremos en este artículo, la clase Number nos sirve para crear objetos que tienen datos numéricos como valor. Es muy probable que no lo llegues a utilizar en ninguna ocasión. Por lo menos en la mayoría de los scripts, para hacer las cosas más dispares, no vas a utilizar esta clase, no obstante viene bien conocerla.

> **Nota:** conocimos el tipo de datos numérico en el primer manual de javascript. Este nos servía para guardar un valores numéricos sin más. Este objeto modeliza este tipo de datos y la clase en si, ofrece algún método que puede ser útil. Para los cálculos matemáticos y el uso de números en general vamos a utilizar siempre las variables numéricas vistas anteriormente.

El valor del objeto Number que se crea depende de lo que reciba el constructor de la clase Number. Con estas reglas:

- Si el constructor recibe un número, entonces inicializa el objeto con el número que recibe. Si recibe un número entrecomillado lo convierte a valor numérico, devolviendo también dicho número.

- Devuelve 0 en caso de que no reciba nada.

- En caso de que reciba un valor no numérico devuelve NaN, que significa "Not a Number" (No es un número)

- Si recibe false se inicializa a 0 y si recibe true se inicializa a 1.

Su funcionamiento se puede resumir en estos ejemplos.

```
var n1 = new Number()
document.write(n1 + "<br>")
//muestra un 0
```

```
var n2 = new Number("hola")
document.write(n2 + "<br>")
//muestra NaN

var n3 = new Number("123")
document.write(n3 + "<br>")
//muestra 123

var n4 = new Number("123asdfQWERTY")
document.write(n4 + "<br>")
//muestra NaN

var n5 = new Number(123456)
document.write(n5 + "<br>")
//muestra 123456

var n6 = new Number(false)
document.write(n6 + "<br>")
//muestra 0
```

```
var n7 = new Number(true)
document.write(n7 + "<br>")
//muestra 1
```

Este ejemplo y el siguiente, se pueden ver en una página a parte.

2.10.1.- Propiedades de la clase Number

Esta clase también nos ofrece varias propiedades que contienen los siguientes valores:

NaN
Como hemos visto, significa Not a Number, o en español, no es un número.

MAX_VALUE y MIN_VALUE
Guardan el valor del máximo y el mínimo valor que se puede representar en Javascript

NEGATIVE_INFINITY y POSITIVE_INFINITY
Representan los valores, negativos y positivos respectivamente, a partir de los cuales hay desbordamiento.

Estas propiedades son de clase, así que accederemos a ellas a partir del nombre de la clase, tal como podemos ver en este ejemplo en el que se muestra cada uno de sus valores.

```
document.write("Propiedad NaN: " + Number.NaN)
document.write("<br>")
document.write("Propiedad MAX_VALUE: " + Number.MAX_VALUE)
document.write("<br>")
document.write("Propiedad MIN_VALUE: " + Number.MIN_VALUE)
document.write("<br>")
document.write("Propiedad NEGATIVE_INFINITY: " + Number.NEGATIVE_INFINITY)
document.write("<br>")
document.write("Propiedad POSITIVE_INFINITY: " + Number.POSITIVE_INFINITY)
```

Los dos ejemplos de este artículo se pueden ver en funcionamiento en una página a parte.

Artículo por Miguel Angel Alvarez

2.11.- Clase Boolean en Javascript

Otra de las clases incorporadas en Javascript, en este caso para crear valores boleanos a partir de valores no boleanos.

En este artículo presentaremos otra de las clases nativas de Javascript, que es la clase Boolean. Esta clase nos sirve para crear valores boleanos y fue añadida en la versión 1.1 de Javascript (con Netscape Navigator 3).

Una de sus posibles utilidades es la de conseguir valores boleanos a partir de datos de cualquier otro tipo. No obstante, al igual que ocurría con la clase Number, es muy probable que no la llegues a utilizar nunca.

> **Nota:** conocimos el tipo de datos boolean en el primer manual de Javascript. Este nos servía para guardar un valor verdadero (true) o falso (false). Esta clase modeliza ese tipo de datos para crear objetos boleanos.

Dependiendo de lo que reciba el constructor de la clase Bolean el valor del objeto boleano que se crea será verdadero o falso, de la siguiente manera

- **Se inicializa a false** cuando no pasas ningún valor al constructor, o si pasas una cadena vacía, el número 0 o la palabra false sin comillas.

- **Se inicializa a true** cuando recibe cualquier valor entrecomillado o cualquier número distinto de 0.

Se puede comprender el funcionamiento de este objeto fácilmente si examinamos unos ejemplos.

```
var b1 = new Boolean()
document.write(b1 + "<br>")
//muestra false
```

```
var b2 = new Boolean("")
document.write(b2 + "<br>")
//muestra false

var b25 = new Boolean(false)
document.write(b25 + "<br>")
//muestra false

var b3 = new Boolean(0)
document.write(b3 + "<br>")
//muestra false

var b35 = new Boolean("0")
document.write(b35 + "<br>")
//muestra true

var b4 = new Boolean(3)
document.write(b4 + "<br>")
//muestra true

var b5 = new Boolean("Hola")
document.write(b5 + "<br>")
//muestra true
```

Se puede ver en funcionamiento el ejemplo en una página a parte.

Artículo por Miguel Angel Alvarez

Parte 3:
Introducción a la programación orientada a objetos en Javascript

Explicamos de manera breve las posibilidades de programación orientada a objetos (POO) disponible en el lenguaje Javascript.

3.1.- Creación de clases en Javascript

Ahora que ya sabemos lo que son los objetos, vamos a ver cómo podemos crear nuestros propios objetos en Javascript.

En capítulos anteriores de esta segunda parte del Manual de Javascript vimos las generalidades sobre la programación orientada a objetos. Además, estuvimos dando un repaso bastante completo a los distintos objetos nativos de Javascript.

Ahora que ya hemos conocido un poco los objetos y hemos aprendido a manejarlos podemos pasar a un tema más avanzado, como es el de construir nuestros propios objetos, que puede sernos útil en ciertas ocasiones para temas avanzados.

Así que vamos a ver cómo podemos definir nuestros propios objetos, con sus propiedades y métodos, de manera que aprendamos el mecanismo, pero sin entrar demasiado en aspectos prácticos que los dejamos para ejemplos del futuro.

Para crear nuestros propios objetos debemos crear una clase, que recordamos que es algo así como la definición de un objeto con sus propiedades y métodos. Para crear la clase en Javascript debemos escribir una función especial, que se encargará de construir el objeto en memoria e inicializarlo. Esta función se le llama constructor en la terminología de la programación orientada a objetos.

```
function MiClase (valor_inicializacion){
    //Inicializo las propiedades y métodos
    this.miPropiedad = valor_inicializacion
    this.miMetodo = nombre_de_una_funcion_definida
}
```

Eso era un constructor. Utiliza la palabra this para declarar las propiedades y métodos del objeto que se está construyendo. This hace referencia al objeto que se está construyendo, pues recordemos que a esta función la llamaremos para construir un objeto. A ese objeto que se está construyendo le vamos asignando valores en sus propiedades y también le vamos asignando nombres de funciones definidas para sus métodos. Al construir un objeto técnicamente es lo mismo declarar una propiedad o un método, solo difiere en que a una propiedad le asignamos un valor y a un método le asignamos una función.

3.1.1.- La clase AlumnoUniversitario

Lo veremos todo más detenidamente si hacemos un ejemplo. En este ejemplo vamos a crear un objeto estudiante universitario. Como estudiante tendrá unas características como el nombre, la edad o el número de matrícula. Además podrá tener algún método como por ejemplo matricular al alumno.

3.1.2.- Constructor: Colocamos propiedades

Veamos cómo definir el constructor de la clase Alumnouniversitario, pero solamente vamos a colocar por ahora las propiedades de la clase.

```
function AlumnoUniversitario(nombre, edad){
    this.nombre = nombre
    this.edad = edad
    this.numMatricula = null
}
```

Los valores de inicialización los recibe el constructor como parámetros, en este caso es sólo el nombre y la edad, porque el número de matrícula no lo recibe un alumno hasta que es matriculado. Es por ello que asignamos a null la propiedad numMatrícula.

En el siguiente artículo explicaremos cómo se pueden realizar métodos y asociarlos a las clases que estamos definiendo.

Artículo por Miguel Angel Alvarez

3.2.- Creación de clases en Javascript II

Aprendemos a construir métodos y asociarlos a nuestros propios objetos de Javascript. También repasamos cómo instanciar nuestros objetos a partir de las definiciones de la clase.

En el artículo anterior del presente manual de Javascript II empezamos a explicar el proceso de creación de una clase. Ahora vamos a continuar con esa definición de la clase para incorporar métodos.

Para construir un método debemos crear una función. Una función que se construye con intención de que sea un método para una clase puede utilizar también la variable this, que hace referencia al objeto sobre el que invocamos el método. Pues debemos recordar que para llamar a un método debemos tener un objeto y this hace referencia a ese objeto.

```
function matriculate(num_matricula){
    this.numMatricula = num_matricula
}
```

La función matricular recibe un número de matrícula por parámetro y lo asigna a la propiedad numMatricula del objeto que recibe este método. Así rellenamos el la propiedad del objeto que nos faltaba.

Vamos a construir otro método que imprime los datos del alumno.

```
function imprimete(){
    document.write("Nombre: " + this.nombre)
    document.write("<br>Edad: " + this.edad)
    document.write("<br>Número de matrícula: " + this.numMatricula)
```

```
}
```

Esta función va imprimiendo todas las propiedades del objeto que recibe el método.

3.2.1.- Constructor: Colocamos métodos

Para colocar un método en una clase debemos asignar la función que queremos que sea el método al objeto que se está creando. Veamos cómo quedaría el constructor de la clase AlumnoUniversitario con el método matricular.

```
function AlumnoUniversitario(nombre, edad){
    this.nombre = nombre
    this.edad = edad
    this.numMatricula = null
    this.matriculate = matriculate
    this.imprimete = imprimete
}
```

Vemos que en las últimas líneas asignamos a los métodos los nombres de las funciones que contienen su código.

3.2.2.- Para instanciar un objeto

Para instanciar objetos de la clase AlumnoUniversitario utilizamos la sentencia new, que ya hemos tenido ocasión de ver en otras ocasiones.

```
miAlumno = new AlumnoUniversitario("José Díaz",23)
```

Este ejemplo lo vamos a completar en el siguiente artículo.

Artículo por Miguel Angel Alvarez

3.3.- Creación de clases en Javascript III

Para ilustrar el modo de trabajo con clases y objetos presentamos el ejemplo completo, en el que creamos una clase, instanciamos objetos y los utilizamos.

En los dos artículos anteriores hemos comenzado una práctica sobre programación orientada a objetos en Javascript que vamos a completar con este texto. Por tanto, tenemos que leer antes de esto el artículo donde comenzamos la clase AlunnoUniversitario y el artículo donde incorporamos los primeros métodos.

Para ilustrar el trabajo con objetos y terminar con el ejemplo del AlunnoUniversitario, vamos a ver todo este proceso en un solo script en el que definiremos la clase y luego la utilizaremos un poquito.

```
//definimos el método matricularte para la clase AlumnoUniversitario
function matriculate(num_matricula){
    this.numMatricula = num_matricula
}
```

```
//definimos el método imprimete para la clase AlumnoUniversitario
function imprimete(){
   document.write("<br>Nombre: " + this.nombre)
   document.write("<br>Edad: " + this.edad)
   document.write("<br>Número de matrícula: " + this.numMatricula)
}

//definimos el constructor para la clase
function AlumnoUniversitario(nombre, edad){
   this.nombre = nombre
```

```
    this.edad = edad
    this.numMatricula = null
    this.matriculate = matriculate
    this.imprimete = imprimete
}

//creamos un alumno
miAlumno = new AlumnoUniversitario("José Díaz",23)

//le pedimos que se imprima
miAlumno.imprimete()

//le pedimos que se matricule
miAlumno.matriculate(305)

//le pedimos que se imprima de nuevo (con el número de matricula relleno)
miAlumno.imprimete()
```

Si lo deseamos, podemos ver el script en funcionamiento en una página a parte.

No vamos a hablar más por el momento sobre cómo crear y utilizar nuestros propios objetos, pero en el futuro trataremos este tema con más profundidad y haremos algún ejemplo adicional.

Artículo por Miguel Angel Alvarez

Parte 4:
Los objetos del navegador: DOM de la página

Comenzamos a trabajar con los objetos que nos sirven para controlar directamente los elementos de la página, los objetos que se generan automáticamente en el navegador al visitar una página. A lo largo de estos artículos trataremos diversos componentes del DOM de Javascript (Modelo de Objetos del Documento).

4.1.- Jerarquía de objetos del navegador

Son los objetos que están disponibles en Javascript para controlar cualquier elemento presente en la página web.

Llegamos al tema más importante para aprender a manejar Javascript con toda su potencia, el tema en el que aprenderemos a controlar al navegador y los distintos elementos de la página.

Sin duda, este tema le va a dar mucha vida a nuestros ejemplos, ya que hasta ahora no tenían mucho carácter práctico porque no trabajaban con el navegador y las páginas, que es realmente para lo que está hecho Javascript. De modo que esperamos que a partir de aquí el manual sea más entretenido para todos, porque va a cubrir los aspectos más prácticos.

Cuando se carga una página, el navegador crea una jerarquía de objetos en memoria que sirven para controlar los distintos elementos de dicha página. Con Javascript y la nomenclatura de objetos que hemos aprendido, podemos trabajar con esa jerarquía de objetos, acceder a sus propiedades e invocar sus métodos.

Cualquier elemento de la página se puede controlar de una manera u otra accediendo a esa jerarquía. Es crucial conocerla bien para poder controlar perfectamente las páginas web con Javascript o cualquier otro lenguaje de programación del lado del cliente.

4.1.1.- Ejemplo de acceso a la jerarquía

Antes de empezar a ver rigurosamente la jerarquía de objetos del navegador, vamos a ver el ejemplo típico de acceso a una propiedad de esta jerarquía para cambiar el aspecto de la página. Se trata de una propiedad de la página que almacena el color de fondo de la página web: la propiedad bgColor del objeto document.

```
document.bgColor = "red"
```

Si ejecutamos esta sentencia en cualquier momento cambiamos el color de fondo de la página a rojo. Hay que fijarse en que la propiedad bgColor tiene la "C" en mayúscula. Es un error típico equivocarse con las mayúsculas y minúsculas en la jerarquía. Si no lo escribimos bien no funcionará y en algunos casos ni siquiera dará un mensaje de error.

En esta página definida con color de fondo blanco hemos cambiado esa propiedad luego con Javascript, por lo que saldrá con color de fondo rojo.

```
<HTML>
<HEAD>
    <TITLE>Prueba bgColor</TITLE>
</HEAD>
<BODY bgcolor=white>

<script>
   document.bgColor = "red"
</script>
</BODY>
</HTML>
```

Podemos ver esta página en marcha en una ventana a parte.

En los ejemplos que hemos visto hasta ahora también hemos hecho uso de los objetos de la jerarquía del navegador. En concreto hemos utilizado mucho el método write() del objeto document para escribir un texto en la página.

```
document.write("El texto a escribir")
```

Artículo por Miguel Angel Alvarez

4.2.- Trabajando con la Jerarquía en Javascript

Vemos por encima la jerarquía entera, detallando alguno de sus elementos y una explicación sobre como se accede a ellos.

Vamos a ver ahora como está compuesta esta jerarquía. Los objetos que forman parte de ella están representados en el gráfico siguiente.

Jerarquía de objetos del navegador en Javascript 1.2.
Podría faltar por recoger algún objeto, pero sirve perfectamente para hacerse una idea de cómo se organizan los objetos en la jerarquía.

Como se puede apreciar, todos los objetos comienzan en un objeto que se llama window. Este objeto ofrece una serie de métodos y propiedades para controlar la ventana del navegador. Con ellos podemos controlar el aspecto de la ventana, la barra de estado, abrir ventanas secundarias y otras cosas que veremos más adelante cuando expliquemos con detalle el objeto.

Además de ofrecer control, el objeto window da acceso a otros objetos como el documento (La página web que se está visualizando), el historial de páginas visitadas o los distintos frames de la ventana. De modo que para acceder a cualquier otro objeto de la jerarquía deberíamos empezar por el objeto window. Tanto es así que javascript entiende perfectamente que la jerarquía empieza en window aunque no lo señalemos.

En los ejemplos incluidos en el capítulo anterior podíamos haber escrito también las sentencias de acceso a la jerarquía empezando por el objeto window, de esta manera.

```
window.document.bgColor = "red"
window.document.write("El texto a escribir")
```

No lo hicimos por que quedase más claro el código y ahorrar algo de texto, pero ahora ya sabemos que toda la jerarquía empieza en el objeto window.

4.2.1.- Las propiedades de un objeto pueden ser a su vez otros objetos

Muchas de las propiedades del objeto window son a su vez otros objetos. Es el caso de objetos como el historial de páginas web o el objeto documento, que tienen a su vez otras propiedades y métodos.

Entre ellos destaca el objeto document, que contiene todas las propiedades y métodos necesarios para controlar muchos aspectos de la página. Ya hemos visto alguna propiedad como bgColor, pero tiene muchas otras como el título de la página, las imágenes que hay incluidas, los formularios, etc. Muchas propiedades de este objeto son a su vez otros objetos, como los formularios. Los veremos todos cuando tratemos cada uno de los objetos por separado. Además, el objeto document tiene métodos para escribir en la página web y para manejar eventos de la página.

4.2.2.- Navegación a través de la jerarquía

El objetivo de este capítulo sobre la jerarquía de objetos es aprender a navegar por ella para acceder a cualquier elemento de la página. Esta no es una tarea difícil, pero puede haber algún caso especial en el que acceder a los elementos de la página se haga de una manera que aun no hemos comentado.

Como ya dijimos, toda la jeraquía empieza en el objeto window, aunque no era necesario hacer referencia a window para acceder a cualquier objeto de la jerarquía. Luego en importancia está el objeto document, donde podemos encontrar alguna propiedad especial que merece la pena comentar por separado, porque su acceso es un poco diferente. Se trata de las propiedades que son arrays, por ejemplo la propiedad images es un array con todas las imágenes de la página web. También encontramos arrays para guardar los enlaces de la página, los applets, los formularios y las anclas.

Cuando una página se carga, el navegador construye en memoria la jerarquía de objetos. De manera adicional, construye también estos arrays de objetos. Por ejemplo, en el caso de las imágenes, va creando el array colocando en la posición 0 la primera imagen, en la posición 1 la segunda imagen y así hasta que las introduce todas. Vamos a ver un bucle que recorre todas las imágenes de la página e imprime su propiedad src, que contiene la URL donde está situada la imagen.

```
for (i=0;i<document.images.length;i++){
    document.write(document.images[i].src)
    document.write("<br>")
}
```

Utilizamos la propiedad length del array images para limitar el número de iteraciones del bucle. Luego utilizamos el método write() del objeto document pasándole el valor de cada una de las propiedades src de cada imagen.

Podemos ver una página con varias imágenes donde se accede a sus propiedades con el bucle anterior.

Ahora vamos a ver el uso de otro array de objetos. En este caso vamos a acceder un poco más dentro de la jerarquía

para llegar a la matriz elements de los objetos formulario, que contiene cada uno de los elementos que componen el formulario. Para ello tendremos que acceder a un formulario de la página, al que podremos acceder por el array de formularios, y dentro de él a la propiedad elements, que es otro array de objetos. Para cada elemento del formulario vamos a escribir su propiedad value, que corresponde con la propiedad value que colocamos en HTML.

```
for (i=0;i<document.forms[0].elements.length;i++){
    document.write(document. forms[0].elements[i].value)
    document.write("<br>")
}
```

Es un bucle muy parecido al que teníamos para recorrer las imágenes, con la diferencia que ahora recorremos el vector de elements, al que accedemos por la jerarquía de objetos pasando por el array de formularios en su posición 0, que corresponde con el primer formulario de la página.

Para ver este ejemplo en funcionamiento, tenemos una página con un formulario donde se ejecuta este recorrido a sus elementos.

Con esto hemos aprendido a movernos por la jerarquía de objetos, con lo que podremos acceder a cualquier elemento del navegador o la página. En adelante conoceremos con detalle cada uno de los objetos de la jerarquía, empezando por el objeto window y bajando por la jerarquía hasta verlos todos.

Artículo por Miguel Angel Alvarez

4.3.- Objeto window de Javascript

Estudiamos el objeto window de Javascript que nos sirve para controlar la ventana del navegador. Detallamos sus propiedades y hacemos un ejemplo.

Es el objeto principal en la jerarquía y contiene las propiedades y métodos para controlar la ventana del navegador. De él dependen todos los demás objetos de la jerarquía. Vamos a ver la lista de sus propiedades y métodos.

4.3.1.- Propiedades del objeto window

A continuación podemos ver las propiedades del objeto window. Hay algunas muy útiles y otras que lo son menos.

closed
Indica la posibilidad de que se haya cerrado la ventana. (Javascript 1.1)

defaultStatus
Texto que se escribe por defecto en la barra de estado del navegador.

document
Objeto que contiene el la página web que se está mostrando.

Frame
Un objeto frame de una página web. Se accede por su nombre.

frames array
El vector que contiene todos los frames de la página. Se accede por su índice a partir de 0.

history
Objeto historial de páginas visitadas.

innerHeight
Tamaño en pixels del espacio donde se visualiza la página, en vertical. (Javascript 1.2)

innerWidth

Tamaño en pixels del espacio donde se visualiza la página, en horizontal. (Javascript 1.2)

length
Numero de frames de la ventana.

location
La URL del documento que se está visualizando. Podemos cambiar el valor de esta propiedad para movernos a otra página. Ver también la propiedad location del objeto document.

locationbar
Objeto barra de direcciones de la ventana. (Javascript 1.2)

menubar
Objeto barra de menús de la ventana. (Javascript 1.2)

name
Nombre de la ventana. Lo asignamos cuando abrimos una nueva ventana.

opener
Hace referencia a la ventana de navegador que abrió la ventana donde estamos trabajando. Se verá con detenimiento en el tratamiento de ventanas con Javascript.

outherHeight
Tamaño en pixels del espacio de toda la ventana, en vertical. Esto incluye las barras de desplazamiento, botones, etc. (Javascript 1.2)

outherWidth
Tamaño en pixels del espacio de toda la ventana, en horizontal. Esto incluye las barras de desplazamiento. (Javascript 1.2)

parent
Hace referencia a la ventana donde está situada el frame donde estamos trabajando. La veremos con detenimiento al estudiar el control de frames con Javascript.

personalbar
Objeto barra personal del navegador. (Javascript 1.2)

self
Ventana o frame actual.

scrollbars
Objeto de las barras de desplazamiento de la ventana.

status
Texto de la barra de estado.

statusbar
Objeto barra de estado del navegador. (Javascript 1.2)

toolbar
Objeto barra de herramientas. (Javascript 1.2)

top
Hace referencia a la ventana donde está situada el frame donde estamos trabajando. Como la propiedad parent.

window
Hace referencia a la ventana actual, igual que la propiedad self.

Vamos a ver un ejemplo de utilización de la propiedad status del objeto window. Esta propiedad sirve para escribir un texto en la barra de estado del navegador (la barra de debajo de la ventana). En este ejemplo hemos tenido que adelantarnos un poco en la marcha del manual, pues utilizamos un manejador de eventos y no hemos visto todavía lo que son. En concreto utilizamos el manejador de eventos onclick, que sirve para ejecutar sentencias Javascript cuando el

usuario pulsa un elemento de la página.

Los manejadores de eventos se colocan en etiquetas HTML, en nuestro caso lo colocamos en un botón de formulario. Las sentencias Javascript asociadas al evento onclick del botón se ejecutarán cuando pulsemos el botón.

Veamos ya el código que hace que se cambie el texto de la barra de estado cuando pulsemos un botón.

```
<form>
<input type="Button" value="Pulsame!" onclick="window.status='Hola a todo el mundo!'">
</form>
```

Simplemente asignamos un texto a la propiedad status del objeto window. El texto que colocamos en la barra de estado está escrito entre comillas simples porque estamos escribiendo dentro de unas comillas dobles.

Podemos ver una página a parte con este ejemplo.

Artículo por Miguel Angel Alvarez

4.4.- Métodos de window en Javascript

El objeto window de Javascript tiene a disposición de los programadores una larga lista de métodos. Los estudiamos y vemos ejemplos.

Vamos a ver ahora los distintos métodos que tiene el objeto window. Muchos de estos métodos habrá que verlos por separado porque son muy útiles y aun no los hemos utilizado, ahora vamos a listarlos y ya veremos algunos ejemplos.

alert(texto)
Presenta una ventana de alerta donde se puede leer el texto que recibe por parámetro

back()
Ir una página atrás en el historial de páginas visitadas. Funciona como el botón de volver de la barra de herramientas. (Javascript 1.2)

blur()
Quitar el foco de la ventana actual. (Javascript 1.1)

captureEvents(eventos)
Captura los eventos que se indiquen por parámetro (Javascript 1.2).

clearInterval()
Elimina la ejecución de sentencias asociadas a un intervalo indicadas con el método setInterval().(Javascript 1.2)

clearTimeout()
Elimina la ejecución de sentencias asociadas a un tiempo de espera indicadas con el método setTimeout().

close()
Cierra la ventana. (Javascript 1.1)

confirm(texto)
Muestra una ventana de confirmación y permite aceptar o rechazar.

find()
Muestra una ventanita de búsqueda. (Javascript 1.2 para Netscape)

focus()
Coloca el foco de la aplicación en la ventana. (Javascript 1.1)

forward()
Ir una página adelante en el historial de páginas visitadas. Como si pulsásemos el botón de adelante del navegador.

(Javascript 1.2)

home()
Ir a la página de inicio que haya configurada en el explorador. (Javascript 1.2)

moveBy(pixelsX, pixelsY)
Mueve la ventana del navegador los pixels que se indican por parámetro hacia la derecha y abajo. (Javascript 1.2)

moveTo(pixelsX, pixelsY)
Mueve la ventana del navegador a la posición indicada en las coordenadas que recibe por parámetro. (Javascript 1.2)

open()
Abre una ventana secundaria del navegador. Se puede aprender a utilizarla en el reportaje de cómo abrir ventanas secundarias.

print()
Como si pulsásemos el botón de imprimir del navegador. (Javascript 1.2)

prompt(pregunta,inicializacion_de_la_respuesta)
Muestra una caja de diálogo para pedir un dato. Devuelve el dato que se ha escrito.

releaseEvents(eventos)
Deja de capturar eventos del tipo que se indique por parámetro. (Javascript 1.2)

resizeBy(pixelsAncho,pixelsAlto)
Redimensiona el tamaño de la ventana, añadiendo a su tamaño actual los valores indicados en los parámetros. El primero para la altura y el segundo para la anchura. Admite valores negativos si se desea reducir la ventana. (Javascript 1.2)

resizeTo(pixelsAncho,pixelsAlto)
Redimensiona la ventana del navegador para que ocupe el espacio en pixels que se indica por parámetro (Javascript 1.2)

routeEvent()
Enruta un evento por la jerarquía de eventos. (Javascript 1.2)

scroll(pixelsX,pixelsY)
Hace un scroll de la ventana hacia la coordenada indicada por parámetro. Este método está desaconsejado, pues ahora se debería utilizar scrollTo()(Javascript 1.1)

scrollBy(pixelsX,pixelsY)
Hace un scroll del contenido de la ventana relativo a la posición actual. (Javascript 1.2)

scrollTo(pixelsX,pixelsY)
Hace un scroll de la ventana a la posición indicada por el parámetro. Este método se tiene que utilizar en lugar de scroll. (Javascript 1.2)

setInterval()
Define un script para que sea ejecutado indefinidamente en cada intervalo de tiempo. (Javascript 1.2)

setTimeout(sentencia,milisegundos)
Define un script para que sea ejecutado una vez después de un tiempo de espera determinado.

stop()
Como pulsar el botón de stop de la ventana del navegador. (Javascript 1.2)

Para ilustrar un poco mejor el funcionamiento de alguno de estos métodos -los más extraños-, hemos creado una página web que los utiliza. El código de la página se muestra a continuación y también podemos ver la página en marcha.

```
<form>
<input type="button" value="Ventana de búsqueda (Solo Netscape)" onClick="window.find()">
<br>
<br>
<input type="button" value="Mover la ventana 10 derecha,10 abajo" onClick="moveBy(10, 10)">
```

```
<br>
<br>
<input type="button" value="Mover la ventana al punto (100,10)" onClick="moveTo(100, 10)">
<br>
<br>
<input type="button" value="Imprimir esta página" onClick="print()">
<br>
<br>
<input type="button" value="Aumenta la ventana 10 ancho,10 largo" onClick="resizeBy(10, 10)">
<br>
<br>
<input type="button" value="Fija el tamaño de la ventana en 400 x 200" onClick="resizeTo(400, 200)">

<br>
<br>
<input type="button" value="Scroll arriba del todo" onClick="scroll(0,0)">
<br>
<br>
<input type="button" value="Scroll arriba 10 pixels" onClick="scrollBy(0,-10)">
</form>
```

Estos ejemplos son muy simples, aunque poco prácticos. Únicamente se trata de una serie de botones que, al pulsarlos, llaman a otros tantos métodos del objeto window. En el atributo onclick de la etiqueta del botón se indican las sentencias Javascript que queremos que se ejecuten cuando se pulsa sobre dicho botón.

En el capítulo siguiente veremos otros ejemplos realizados con métodos del objeto window de Javascript, un poco más detallados.

Artículo por Miguel Angel Alvarez

4.5.- Ejemplos de métodos de window

Otros ejemplos de métodos del objeto window de Javascript, relatados con detalle.

Ahora vamos a realizar algún ejemplo de utilización de los métodos de la ventana. Nos vamos a centrar en los ejemplos que sirven para sacar cajas de diálogo, que son muy útiles.

4.5.1.- Caja de alerta

Para sacar un texto en una ventanita con un botón de aceptar. Recibe el texto por parámetro.

```
window.alert("Este es el texto que sale")
```

Como el objeto window se sobreentiende podemos escribirlo así.

```
alert("Este es el texto que sale")
```

Saca una ventana como la que se puede ver en esta página.

4.5.2.- Caja de confirmación

Muestra una ventana con un texto indicado por parámetro y un botón de aceptar y otro de rechazar. Dependiendo del botón que se pulsa devuelve un true (si se pulsa aceptar) o un false (si se pulsa rechazar).

```
<script>
var respuesta = confirm("Aceptame o rechazame")
alert ("Has pulsado: " + respuesta)
</script>
```

Este script muestra una caja de diálogo confirm y luego muestra en otra caja de diálogo alert el contenido de la variable

que devolvió la caja de diálogo. Nuevamente, podemos ver el funcionamiento de este script si accedemos a esta página a parte.

4.5.3.- Caja de introducción de un dato

Muestra una caja de diálogo donde se formula una pregunta y hay un campo de texto para escribir una respuesta. El campo de texto aparece relleno con lo que se escriba en el segundo parámetro del método. También hay un botón de aceptar y otro de cancelar. En caso de pulsar aceptar, el método devuelve el texto que se haya escrito. Si se pulsó cancelar devuelve null.

El ejemplo siguiente sirve para pedir el nombre de la persona que está visitando la página y luego mostrar en la página un saludo personalizado. Utiliza un bucle para repetir la toma de datos siempre que el nombre de la persona sea null (porque pulsó el botón de cancelar) o sea un string vacío (porque no escribió nada).

```
<script>
nombre = null
while (nombre == null || nombre == ""){
    nombre = prompt("Dime tu nombre:","")
}
document.write("<h1>Hola " + nombre + "</h1>")
</script>
```

Si nos fijamos en la caja prompt veremos que recibe dos parámetros. El segundo era el texto por defecto que sale en la caja como respuesta. Lo hemos dejado como un string vacío para que no salga nada como texto por defecto.

Podemos ver este último script en funcionamiento en una página a parte.

Hasta aquí los ejemplos de los métodos del objeto window. De todos modos, en el resto del manual tendremos ocasión de ver cómo trabajar con muchas propiedades y métodos de este objeto.

Artículo por Miguel Angel Alvarez

4.6.- Objeto document en Javascript

Una descripción del objeto de Javascript que sirve para controlar el domumento que se visualiza en el navegador. También hay una lista de todas sus propiedades.

Con el objeto document se controla la página web y todos los elementos que contiene. El objeto document es la página actual que se está visualizando en ese momento. Depende del objeto window, pero también puede depender del objeto frame en caso de que la página se esté mostrando en un frame.

4.6.1.- Propiedades del objeto document

Veamos una lista de las propiedades del objeto document y luego veremos algún ejemplo.

alinkColor
Color de los enlaces activos

Anchor
Un ancla de la página. Se consigue con la etiqueta <A name="nombre_del_ancla">. Se accede por su nombre.

anchors array
Un array de las anclas del documento.

Applet
Un applet de la página. Se accede por su nombre. (Javascript 1.1)

applets array
Un array con todos los applets de la página. (Javascript 1.1)

Area
Una etiqueta <AREA>, de las que están vinculadas a los mapas de imágenes (Etiqueta). Se accede por su nombre. (Javascript 1.1)

bgColor
El color de fondo del documento.

classes
Las clases definidas en la declaración de estilos CSS. (Javascript 1.2)

cookie
Una cookie

domain
Nombre del dominio del servidor de la página.

Embed
Un elemento de la pagina incrustado con la etiqueta <EMBED>. Se accede por su nombre. (Javascript 1.1)

embeds array
Todos los elementos de la página incrustados con <EMBED>. (Javascript 1.1)

fgColor
El color del texto. Para ver los cambios hay que reescribir la página.

From
Un formulario de la página. Se accede por su nombre.

forms array
Un array con todos los formularios de la página.

ids
Para acceder a estilos CSS. (Javascript 1.2)

Image
Una imagen de la página web. Se accede por su nombre. (Javascript 1.1)

images array
Cada una de las imágenes de la página introducidas en un array. (Javascript 1.1)

lastModified
La fecha de última modificación del documento.

linkColor
El color de los enlaces.

Link
Un enlace de los de la página. Se accede por su nombre.

links array
Un array con cada uno de los enlaces de la página.

location
La URL del documento que se está visualizando. Es de solo lectura.

referrer
La página de la que viene el usuario.

tags
Estilos definidos a las etiquetas de HTML en la página web. (Javascript 1.2)

title
El titulo de la página.

URL
Lo mismo que location, pero es aconsejable utilizar location ya que URL no existe en todos los navegadores.

vlinkColor
El color de los enlaces visitados.

Artículo por Miguel Angel Alvarez

4.7.- Ejemplos de propiedades de document

Vemos varios ejemplos de acceso y manipulación de las propiedades del objeto document de Javascript.

Después de estudiar las propiedades del objeto document, vamos a ver algún ejemplo para ilustrar el modo de acceso y utilización de las mismas.

4.7.1.- Ejemplo con la propiedad bgColor

Vamos a utilizar el evento onclick para colocar tres botones en la página que al pulsarlos nos cambie el color del fondo de la página.

```
<script>
function cambiaColor(colorin){
    document.bgColor = colorin
}
```

</script>
<form>
<input type="Button" value="Rojo" onclick="cambiaColor('ff0000')">
<input type="Button" value="Verde" onclick="cambiaColor('00ff00')">
<input type="Button" value="Azul" onclick="cambiaColor('0000ff')">
</form>

Primero definimos una función que será la encargada de cambiar el color y luego tres botones que llamarán a la función cuando sean pulsados pasándole el color como parámetro.

Podemos ver el ejemplo en marcha.

4.7.2.- Propiedad title

La propiedad title guarda la cadena que conforma el título de nuestra página web. Si accedemos a dicha propiedad obtendremos el título y si la cambiamos, cambiará el título de la página web.

> **Nota:** recordamos que el título se puede ver en la barra de arriba del todo de la ventana del navegador.

Vamos a mostrar e título de la página en una caja de alerta.

```
alert (document.title)
```

Ahora vamos a hacer una función que modifica el título de la página , asignándole el texto que le llegue por parámetro.

```
function cambiaTitulo(texto){
    document.title = texto
}
```

Como en el ejemplo anterior, vamos a crear varios botones que llamen a la función pasándole distintos textos, que se colocarán en el título de la página.

```
<form>
<input type="Button" value="Titulo = Hola a todos" onclick="cambiaTitulo('Hola a todos')">
<input type="Button" value="Titulo = Bienvenidos a mi página web" onclick="cambiaTitulo('Bienvenidos
a mi página web')">
<input type="Button" value="Titulo = Más dias que longanizas" onclick="cambiaTitulo('Más dias que
longanizas')">
</form>
```

Podemos ver en funcionamiento este script en otra página web.

Artículo por Miguel Angel Alvarez

4.8.- Métodos de document

Vemos una lista de los eventos disponibles en el objeto document.

El objeto document, localizado debajo del objeto window en la jerarquía de objetos de Javascript, también tiene una lista de métodos interesantes. La vemos a continuación.

captureEvents()
Para capturar los eventos que ocurran en la página web. Recibe como parámetro el evento que se desea capturar.

close()
Cierra el flujo del documento. (Se verá más adelante en este manual un artículo sobre el flujo del documento)

contextual()
Ofrece una línea de control de los estilos de la página. En el caso que deseemos especificarlos con Javascript.

getSelection()
Devuelve un string que contiene el texto que se ha seleccionado. Sólo funciona en Netscape.

handleEvent()
Invocas el manejador de eventos del elemento especificado.

open()
Abre el flujo del documento.

releaseEvents()
Liberar los eventos capturados del tipo que se especifique, enviándolos a los objetos siguientes en la jerarquía.

routeEvent()
Pasa un evento capturado a través de la jerarquía de eventos habitual.

write()
Escribe dentro de la página web. Podemos escribir etiquetas HTML y texto normal.

writeln()
Escribe igual que el método write(), aunque coloca un salto de línea al final.

Los eventos de document sirven principalmente para controlar dos cosas. Un grupo nos ofrece una serie de funciones para el control de los documentos, como escribir, abrirlos y cerrarlos. Los veremos en el capítulo siguiente que habla sobre el control del flujo de escritura del documento. El otro grupo ofrece herramientas para el control de los eventos en el documento y lo veremos más adelante cuando tratemos con detenimiento el tema de eventos.

Se nos queda un poco suelto el método getSelection(), que sólo funciona en los navegadores de Netscape y, por tanto, no resulta muy útil para aplicaciones que deseemos que sean compatibles en todos los sistemas. Aun así, haremos el

ejemplo sobre este método, ya que los otros los vamos a ver en siguientes capítulos.

El ejemplo consiste en una página que tiene un poco de texto y un botón. En la página podremos seleccionar algo de texto y luego apretar el botón, que llama a una función que muestra en una caja alert el texto que se ha seleccionado. El código es el siguiente:

```
<html>
<head>
<title>Rescatar lo seleccionado</title>
<script language="JavaScript">
function mostrarSeleccionado(){
    alert("Has seleccionado:n" + document.getSelection())
}
</script>
</head>
```

<body>
<h1>Rescatar lo seleccionado</h1>

<form>
<input type="button" value="pulsame!" onClick="mostrarSeleccionado()">
</form>

Selecciona cualquier texto de la página y pulsa sobre el botón.

</body>
</html>

Se puede ver en funcionamiento el script en una página aparte, aunque sólo funcionará en Netscape e Internet Explorer dará un error.

Artículo por Miguel Angel Alvarez

4.9.- Flujo de escritura del documento

Es el proceso en el que el navegador escribe la página. Para escribir en la página desde Javascript el flujo del documento debe estar abierto.

Acerca del objeto document también es interesante hablar un poco sobre el flujo de escritura del documento o página web.

Cuando el navegador lee una página web la va interpretando y escribiendo en su ventana. El proceso en el que el navegador está escribiendo la página le llamamos flujo de escritura del documento. El flujo comienza cuando se empieza a escribir la página y dura hasta que ésta ha terminado de escribirse. Una vez terminada la escritura de la página el flujo de escritura del documento se cierra automáticamente. Con esto termina la carga de la página.

Una vez cerrado el flujo no se puede escribir en la página web, ni texto ni imágenes ni otros elementos.

En javascript tenemos que respetar el flujo de escritura del documento forzosamente. Es por ello que sólo podemos ejecutar sentencias document.write() (método write() del objeto document) cuando está abierto el flujo de escritura del documento, o lo que es lo mismo, mientras se está cargando la página.

Si recordamos las dos formas de ejecutar un script en Javascript:

- Ejecución de los scripts mientras que carga la página. Aquí podremos ejecutar document.write() y lo hemos hecho habitualmente en los ejemplos anteriores.
- Ejecución de los scripts cuando la página ha sido cargada, como respuesta a un evento del usuario. Aquí no

podemos hacerlo porque la página ha terminado de cargarse, de hecho, no lo hemos hecho nunca hasta ahora.

Hay un matiz que dar a que no se puede escribir en la página cuando ya está cerrado el flujo. En realidad si que se puede, pero necesitamos abrir el flujo otra vez para escribir en la página, tanto es así que aunque nosotros no lo abramos explícitamente Javascript se encargará de ello. Lo que tiene que quedar claro es que si hacemos un document.write() el flujo tiene que estar abierto y si no lo está se abrirá. El problema de abrir el flujo de escritura del documento una vez ya está escrita la página es que se borra todo su contenido.

Para que quede claro vamos a hacer un script para escribir en la página una vez ésta ha sido cargada. Simplemente necesitamos un botón y al pulsarlo ejecutar el método write() del objeto document.

```
<form>
<INPUT type=button value=escribir onclick="window.document.write('hola')">
</form>
```

Si nos fijamos, en el manejador de eventos onclick hemos colocado la jerarquía de objetos desde el principio, es decir, empezando por el objeto window. Esto es porque hay algunos navegadores que no sobreentienden el objeto window en las sentencias escritas en los manejadores de eventos.

Podemos ver el ejemplo en funcionamiento.

El resultado de la ejecución puede variar de un navegador a otro, pero en cualquier caso se borrará la página que se está ejecutando.

4.9.1.- Métodos open() y close() de document

Los métodos open() y close() del objeto document sirven para controlar el flujo del documento desde Javascript. Nos permiten abrir y cerrar el documento explícitamente.

El ejemplo de escritura en la página anterior se debería haber escrito con su correspondiente apertura y cierre del documento y hubiese quedado algo parecido a esto.

```
<script>
function escribe(){
    document.open()
    window.document.write('Hola')
    document.close()
}
</script>
<form>
<INPUT type=button value=escribir onclick="escribe()">
</form>
```

Vemos que ahora no escribimos las sentencias dentro del manejador, porque, cuando hay más de una sentencia, queda más claro poner una llamada a una función y en la función colocamos las sentencias que queramos.

Las sentencias del ejemplo anterior, que cubren la apertura, escritura y cierre del documento. Se pueden ver en marcha aquí.

Artículo por Miguel Angel Alvarez

Parte 5:
Trabajo con formularios en Javascript

Los formularios forman parte del DOM de la página y son un grupo de elementos con los que trabajaremos muy habitualmente y que por tanto, merece la pena prestar especial atención. Aprenderemos a recibir y alterar dinámicamente los valores y estados de los elementos de formulario con Javascript.

5.1.- Trabajo con formularios en Javascript

Para continuar vamos a ver una serie de capítulos enfocados a aprender a trabajar con los formularios. Ahora veremos como acceder a los formularios y sus elementos.

Para continuar vamos a ver una serie de capítulos enfocados a aprender a trabajar con los formularios, acceder a sus elementos para controlar sus valores y actualizarlos.

El objeto form depende en la jerarquía de objetos del objeto document. En un objeto form podemos encontrar algunos métodos y propiedades, pero lo más destacado que podremos encontrar son cada uno de los elementos del formulario. Es decir, de un formulario dependen todos los elementos que hay dentro, como pueden ser campos de texto, cajas de selección, áreas de texto, botones de radio, etc.

Para acceder a un formulario desde el objeto document podemos hacerlo de dos formas.

1. A partir de su nombre, asignado con el atributo NAME de HTML.
2. Mediante la matriz de formularios del objeto document, con el índice del formulario al que queremos acceder.

Para este formulario

```
<FORM name="f1">
<INPUT type=text name=campo1>
<INPUT type=text name=campo2>
</FORM>
```

Podremos acceder con su nombre de esta manera.

```
document.f1
```

O con su índice, si suponemos que es el primero de la página.

```
document.forms[0]
```

De similar manera accedemos a los elementos de un formulario, que dependen del objeto form.

1. A partir del nombre del objeto asignado con el atributo NAME de HTML.
2. Mediante la matriz de elementos del objeto form, con el índice del elemento al que queremos acceder.

Podríamos acceder al campo 1 del anterior formulario de dos maneras. Con su nombre lo haríamos así.

```
document.f1.campo1
```

o a partir del array de elementos.

document.f1.elements[0] (utilizamos el índice 0 porque es el primer elemento y en Javascript los arrays empiezan por 0.)

Si deseamos acceder al segundo campo del formulario escribiríamos una de estas dos cosas:

```
document.f1.campo2
document.f1.elements[1]
```

recordamos que también podemos acceder a un fomulario por el array de forms, indicando el índice del formulario al que acceder. De este modo, el acceso al campo2 sería el siguiente:

```
document.forms[0].campo2
document.forms[0].elements[1]
```

En estos casos hemos supuesto que este formulario es el primero que hay escrito en el código HTML, por eso accedemos a él con el índice 0.

Esperamos que haya quedado claro el acceso a formularios y sus campos. Pasaremos entonces, sin más, a un ejemplo para practicar todo esto.

Artículo por Miguel Angel Alvarez

5.2.- Ej. trabajo con formularios. Calculadora sencilla

Vamos a ver un ejemplo del trabajo con formularios en el que desarrollaremos una calculadora sencilla.

Para ilustrar un poco el trabajo con formularios, vamos a realizar un ejemplo práctico. Puede que algunas cosas que vamos a tratar queden un poco en el aire porque no se hayan explicado con detenimiento antes, pero seguro que nos sirve para enterarnos de cómo se trabaja con formularios y las posibilidadedes que tenemos.

5.2.1.- Ejemplo calculadora sencilla

En este ejemplo vamos a construir una calculadora, aunque bastante sencilla, que permita realizar las operaciones básicas. Para hacer la calculadora vamos a realizar un formulario en el que vamos a colocar tres campos de texto, los dos primeros para los operandos y un tercero para el resultado. Además habrán unos botones para hacer las operaciones básicas.

El formulario de la calculadora se puede ver aquí.

```
<form name="calc">
<input type="Text" name="operando1" value="0" size="12">
<br>
<input type="Text" name="operando2" value="0" size="12">
<br>
<input type="Button" name="" value=" + " onclick="calcula('+')">
```

```
<input type="Button" name="" value=" - " onclick="calcula('-')">
<input type="Button" name="" value=" X " onclick="calcula('*')">
<input type="Button" name="" value=" / " onclick="calcula('/')">
<br>
<input type="Text" name="resultado" value="0" size="12">
</form>
```

Mediante una función vamos a acceder a los campos del formulario para recoger los operandos en dos variables. Los campos de texto tienen una propiedad llamada value que es donde podemos obtener lo que tienen escrito en ese momento. Mas tarde nos ayudaremos de la función eval() para realizar la operación. Pondremos por último el resultado en el campo de texto creado en tercer lugar, utilizando también la propiedad value del campo de texto.

A la función que realiza el cálculo (que podemos ver a continuación) la llamamos apretando los botones de cada una de las operaciones. Dichos botones se pueden ver en el formulario y contienen un atributo onclick que sirve para especificar las sentencias Javascript que deseamos que se ejecuten cuando el usuario pulse sobre él. En este caso, la sentencia a ejecutar es una llamada a la función calcula() pasando como parámetro el símbolo u operador de la operación que deseamos realizar. El

5.2.2.- script con la función calcula()

```
<script>
function calcula(operacion){
    var operando1 = document.calc.operando1.value
    var operando2 = document.calc.operando2.value
    var result = eval(operando1 + operacion + operando2)
    document.calc.resultado.value = result
}
</script>
```

La función eval(), recordamos, que recibía un string y lo ejecutaba como una sentencia Javascript. En este caso va a recibir un número que concatenado a una operación y otro número será siempre una expresión aritmética que eval() solucionará perfectamente.

Podemos ver el ejemplo de la calculadora en funcionamiento.

El acceso a otros elementos de los formularios se hace de manera parecida en cuanto respecta a la jerarquía de objetos, aunque como cada elemento tiene sus particularidades las cosas que podremos hacer con ellos diferirán un poco. Lo veremos un poco más adelante.

Artículo por Miguel Angel Alvarez

5.3.- Propiedades y métodos del objeto form

Echamos un vistazo a las distintas propiedades y métodos del objeto form de Javascript. Mostramos algún ejemplo de utilización de propiedades y una sencilla validación de formulario y envío con el método submit().

Vamos a ver ahora el objeto form por si solo, para destacar sus propiedades y métodos.

5.3.1.- Propiedades del objeto form

Tiene unas cuantas propiedades para ajustar sus atributos mediante Javascript.

action
Es la acción que queremos realizar cuando se submite un formulario. Se coloca generalmente una dirección de correo o la URL a la que le mandaremos los datos. Corresponde con el atributo ACTION del formulario.

elements array
La matriz de elementos contiene cada uno de los campos del formulario.

encoding
El tipo de codificación del formulario

length
El número de campos del formulario.

method
El método por el que mandamos la información. Corresponde con el atributo METHOD del formulario.

name
El nombre del formulario, que corresponde con el atributo NAME del formulario.

target
La ventana o frame en la que está dirigido el formulario. Cuando se submita se actualizará la ventana o frame indicado. Corresponde con el atributo target del formulario.

5.3.2.- Ejemplos de trabajo con las propiedades

Con estas propiedades podemos cambiar dinámicamente con Javascript los valores de los atributos del formulario para hacer con él lo que se desee dependiendo de las exigencias del momento.

Por ejemplo podríamos cambiar la URL que recibiría la información del formulario con la instrucción.

```
document.miFormulario.action = "miPágina.asp"
```

O cambiar el target para submitir un formulario en una posible ventana secundaria llamada mi_ventana.

```
document.miFormulario.target = "mi_ventana"
```

5.3.3.- Métodos del objeto form

Estos son los métodos que podemos invocar con un formulario.

submit()
Para hacer que el formulario se submita, aunque no se haya pulsado el botón de submit.

reset()
Para reinicializar todos los campos del formulario, como si se hubiese pulsado el botón de reset. (Javascript 1.1)

5.3.4.- Ejemplo de trabajo con los métodos

Vamos a ver un ejemplo muy sencillo sobre cómo validar un formulario para submitirlo en caso de que esté relleno. Para ello vamos a utilizar el método submit() del formulario.

El mecanismo es el siguiente: en vez de colocar un botón de submit colocamos un botón normal (<INPUT type="button">) y hacemos que al pulsar ese botón se llame a una función que es la encargada de validar el formulario y, en caso de que esté correcto, submitirlo.

El formulario quedaría así.

```
<form name="miFormulario" action="mailto:promocion@guiarte.com" enctype="text/plain">
<input type="Text" name="campo1" value="" size="12">
<input type="button" value="Enviar" onclick="validaSubmite()">
</form>
```

Nos fijamos en que no hay botón de submit, sino un botón normal con una llamada a una función que podemos ver a continuación.

```
function validaSubmite(){
    if (document.miFormulario.campo1.value == "")
       alert("Debe rellenar el formulario")
    else
       document.miFormulario.submit()
}
```

En la función se comprueba si lo que hay escrito en el formulario es un string vacío. Si es así se muestra un mensaje de alerta que informa que se debe rellenar el formulario. En caso de haya algo en el campo de texto submite el formulario utilizando el método submit del objeto form.

Artículo por Miguel Angel Alvarez

5.4.- Control de campos de texto con Javascript

Explicación y documentación de los campos de texto y su control con Javascript. Se incluyen los campos de tipo text, password y hidden.

Vamos a ver ahora los campos donde podemos guardar cadenas de texto, es decir, los campos de texto, password y hidden. Hay otro campo relacionado con la escritura de texto, el campo TextArea, que veremos más adelante.

5.4.1.- Campo Text

Es el campo que resulta de escribir la etiqueta <INPUT type="text">. Lo hemos utilizado hasta el momento en varios ejemplos, pero vamos a parar un momento en él para describir sus propiedades y métodos.

5.4.2.- Propiedades del campo text

Vemos la lista de propiedades de estos tipos de campo.

defaultValue
Es el valor por defecto que tiene un campo. Lo que contiene el atributo VALUE de la etiqueta <INPUT>.

form
Hace referencia al formulario.

name
Contiene el nombre de este campo de formulario

type
Contiene el tipo de campo de formulario que es.

value
El texto que hay escrito en el campo.

Vamos a ver un ejemplo sobre lo que puede hacer la propiedad defaultValue. En este ejemplo tenemos un formulario y un botón de reset. Si pulsamos el botón de reset el campo de texto se vacía porque su value de HTML era un string vacío. Pero si pulsamos el botón siguiente llamamos a una función que cambia el valor por defecto de ese campo de texto, de modo que al pulsar el botón de reset mostrará el nuevo valor por defecto.

Este es el código de la página completa.

```
<html>
<head>
   <title>Cambiar el valor por defecto</title>
   <script>
    function cambiaDefecto(){
```

```
        document.miFormulario.campo1.defaultValue = "Hola!!"
    }
   </script>
</head>
```

```
<body>
<form name="miFormulario" action="mailto:promocion@guiarte.com" enctype="text/plain">
<input type="Text" name="campo1" value="" size="12">
<input type="Reset">
<br>
<br>
<input type="button" value="Cambia valor por defecto" onclick="cambiaDefecto()">
</form>
</body>
</html>
```

Se puede ver en funcionamiento en esta página.

5.4.3.- Métodos del objeto Text

Se pueden invocar los siguientes métodos sobre los objetos tipo Text.

blur()
Retira el foco de la aplicación del campo de texto.

focus()
Pone el foco de la aplicación en el campo de texto.

select()
Selecciona el texto del campo.

Como ejemplo vamos a mostrar una función que selecciona el texto de un campo de texto de un formulario como el de la página del ejemplo anterior. Para hacerlo hemos utilizado dos métodos, el primero para pasar el foco de la aplicación al campo de texto y el segundo para seleccionar el texto.

```
function seleccionaFoco(){
    document.miFormulario.campo1.focus()
    document.miFormulario.campo1.select()
}
```

Puede verse en funcionamiento en esta página.

5.4.4.- Campos Password

Estos funcionan igual que los hidden, con la peculiaridad que el contenido del campo no puede verse escrito en el campo, por lo que salen asteríscos en lugar del texto.

5.4.5.- Campos Hidden

Los campos hidden son como campos de texto que están ocultos para el usuario, es decir, que no se ven en la página. Son muy útiles en el desarrollo de webs para pasar variables en los formularios a las que no debe tener acceso el usuario.

Se colocan en con HTML con la etiqueta <INPUT type=hidden> y se rellenan de datos con su atributo value. Mas tarde podremos cambiar el dato que figura en el campo accediendo a la propiedad value del campo de texto igual que lo hemos hecho antes.

```
document.miFormulario.CampoHidden.value = "nuevo texto"
```

El campo hidden sólo tiene algunas de las propiedades de los campos de texto. En concreto tiene la propiedad value y

las propiedades que son comunes de todos los campos de formulario: name, from y type, que ya se describieron para los campos de texto.

Artículo por Miguel Angel Alvarez

5.5.- Control de Checkbox en Javascript

Capítulo sobre el control del elemento de formulario tipo checkbox o caja de verificación. Estudiamos sus métodos y propiedades, con ejemplos.

Los checkbox son las unas cajas que permiten marcarlas o no para verificar alguna cosa en un formulario. Podemos ver una caja checkbok a continuación.

☐

Los checkbox se consiguen con la etiqueta <INPUT type=checkbox>. Con el atributo NAME de la etiqueta <INPUT> le podemos dar un nombre para luego acceder a ella con javascript. Con el atributo CHECKED indicamos que el campo debe aparecer chequeado por defecto.

Con Javascript, a partir de la jerarquía de objetos del navegador, tenemos acceso al checkbox, que depende del objeto form del formulario donde está incluido.

5.5.1.- Propiedades de un checkbox

Las propiedades que tiene un checkbox son las siguientes.

checked
Informa sobre el estado del checkbox. Puede ser true o false.

defaultChecked
Si está chequeada por defecto o no.

value
El valor actual del checkbox.

También tiene las propiedades form, name, type como cualquier otro elemento de formulario.

5.5.2.- Métodos del checkbox

Veamos la lista de los métodos de un checkbox.

click()
Es como si hiciésemos un click sobre el checkbox, es decir, cambia el estado del checkbox.

blur()
Retira el foco de la aplicación del checkbox.

focus()
Coloca el foco de la aplicación en el checkbox.

Para ilustrar el funcionamiento de las checkbox vamos a ver una página que tiene un checkbox y tres botones. Los dos primeros para mostrar las propiedades checked y value y el tercero para invocar a su método click() con objetivo de simular un click sobre el checkbox.

```
<html>
<head>
```

```
    <title>Ejemplo Checkbox</title>
<script>
function alertaChecked(){
    alert(document.miFormulario.miCheck.checked)
}
function alertaValue(){
    alert(document.miFormulario.miCheck.value)
}
function metodoClick(){
    document.miFormulario.miCheck.click()
}
</script>
</head>
```

```
<body>
<form name="miFormulario" action="mailto:promocion@guiarte.com" enctype="text/plain">
<input type="checkbox" name="miCheck">
<br>
<br>
<input type="button" value="informa de su propiedad checked" onclick="alertaChecked()">
<input type="button" value="informa de su propiedad value" onclick="alertaValue()">
<br>
<br>
<input type="button" value="Simula un click" onclick="metodoClick()">
</form>
</body>
</html>
```

Puede verse la página en funcionamiento aquí.

Artículo por Miguel Angel Alvarez

5.6.- Control de botones de radio en Javascript

Explicación sobre el manejo de radio buttons en Javascript. Lista de métodos y propiedades junto con algún ejemplo de su funcionamiento.

El botón de radio (o radio button en inglés) es un elemento de formulario que permite seleccionar una opción y sólo una, sobre un conjunto de posibilidades. Se puede ver a continuación.

◉ Blanco
○ Rojo
○ Verde

> **Nota:** En la parte de arriba podemos ver tres radio buttons en lugar de uno solo. Se colocan tres botones porque así podemos examinar su funcionamiento al formar parte de un grupo. Veamos que al seleccionar una opción se deselecciona la opción que estuviera marcada antes.

Se consiguen con la etiqueta <INPUT type=radio>. Con el atributo NAME de la etiqueta <INPUT> les damos un nombre para agrupar los radio button y que sólo se pueda elegir una opción entre varias. Con el atributo value indicamos el valor de cada uno de los radio buttons. El atributo checked nos sirve para indicar cuál de los radio butons tiene que estar seleccionado por defecto.

Referencia: Explicamos en detalle la creación de botones de radio en nuestro manual de HTML, en el capítulo Otros elementos de formularios.

Cuando en una página tenemos un conjunto de botones de radio se crea un objeto radio por cada uno de los botones. Los objetos radio dependen del formulario y se puede acceder a ellos por el array de elements, sin embargo también se crea un array con los botones de radio. Este array depende del formulario y tiene el mismo nombre que los botones de radio.

5.6.1.- Propiedades del objeto radio

Veamos una lista de las propiedades de este elemento.

checked
Indica si está chekeado o no un botón de radio.

defaultChecked
Su estado por defecto.

value
El valor del campo de radio, asignado por la propiedad value del radio.

Length (como propiedad del array de radios)
El número de botones de radio que forman parte en el grupo. Accesible en el vector de radios.

5.6.2.- Métodos del objeto radio

Son los mismos que los que tenía el objeto checkbox.

5.6.3.- Ejemplo de utilización

Veamos con un ejemplo el método de trabajo con los radio buttons en el que vamos a colocar un montón de ellos y cada uno tendrá asociado un color. También habrá un botón y al pulsarlo cambiaremos el color de fondo de la pantalla al color que esté seleccionado en el conjunto de botones de radio.

Vamos a ver la página entera y luego la comentamos.

```
<html>
<head>
    <title>Ejemplo Radio Button</title>
<script>
function cambiaColor(){
    var i
    for (i=0;i<document.fcolores.colorin.length;i++){
       if (document.fcolores.colorin[i].checked)
          break;
    }
    document.bgColor = document.fcolores.colorin[i].value
}
</script>
</head>
```

```
<body>
<form name=fcolores>
<input type="Radio" name="colorin" value="ffffff" checked> Blanco
<br>
<input type="Radio" name="colorin" value="ff0000"> Rojo
<br>
<input type="Radio" name="colorin" value="00ff00"> Verde
```

```
<br>
<input type="Radio" name="colorin" value="0000ff"> Azul
<br>
<input type="Radio" name="colorin" value="ffff00"> Amarillo
<br>
<input type="Radio" name="colorin" value="00ff00"> Turquesa
<br>
<input type="Radio" name="colorin" value="ff00ff"> Morado
<br>
<input type="Radio" name="colorin" value="000000"> Negro
<br>
<br>
<input type="Button" name="" value="Cambia Color" onclick="cambiaColor()">
</form>
</body>
</html>
```

Primero podemos fijarnos en el formulario y en la lista de botones de radio. Todos se llaman "colorin", así que están asociados en un mismo grupo. Además vemos que el atributo value de cada botón cambia. También vemos un botón abajo del todo.

Con esta estructura de formulario tendremos un array de elements de 9 elementos, los 8 botones de radio y el botón de abajo.

Además tendremos un array de botones de radio que se llamará colorín y depende del formulario, accesible de esta manera.

```
document.form.colorin
```

Este array tiene en cada posición uno de los botones de radio. Así en la posición 0 está el botón del color blanco, en la posición 1 el del color rojo... Para acceder a esos botones de radio lo hacemos con su índice.

```
document.fcolores.colorin[0]
```

Si queremos acceder por ejemplo a la propiedad value del último botón de radio escribimos lo siguiente.

```
document.fcolores.colorin[7].value
```

La propiedad length del array de radios nos indica el número de botones de radio que forman parte del grupo.

```
document.fcolores.colorin.length
```

En este caso la propiedad length valdrá 8.

Con estas notas podremos entender más o menos bien la función que se encarga de encontrar el radio button seleccionado y cambiar el color de fondo de la página.

Se define una variable en la que introduciremos el índice del radio button que tenemos seleccionado. Para ello vamos recorriendo el array de botones de radio hasta que encontramos el que tiene su propiedad checked a true. En ese momento salimos del bucle, con lo que la variable i almacena el índice del botón de radio seleccionado. En la última línea cambiamos el color de fondo a lo que hay en el atributo value del radio button seleccionado.

Podemos ver ese ejemplo en funcionamiento.

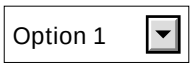
Referencia: Si deseamos profundizar el el control de botones de radio podemos acceder a un taller relacionado: Inhibir radio button con Javascript

Artículo por Miguel Angel Alvarez

5.7.- Control de campos select con Javascript

Descripción y ejemplo para el control de campos de formulario select, también llamados listas desplegables o combo box.

El objeto select de un formulario es una de esas listas desplegables que nos permiten seleccionar un elemento. Se despliegan apretando sobre una flecha, a continuación se puede escoger un elemento y para acabar se vuelven a plegar. Se puede ver un elemento select de un formulario a continuación.



Uno de estos elementos se puede obtener utilizando la etiqueta <SELECT> dentro de un formulario. A la etiqueta le podemos añadir un atributo para darle el nombre, NAME, para luego acceder a ese campo mediante Javascript. Para expresar cada una de las posibles opciones del campo select utilizamos una etiqueta <OPTION> a la que le introducimos un atributo VALUE para expresar su valor. El texto que colocamos después de la etiqueta <OPTION> sirve como etiqueta de esa opción, es el texto que ve el usuario asociado a esa opción.

5.7.1.- Propiedades del objeto select

Vamos a ver una lista de las propiedades de este elemento de formulario.

length
Guarda la cantidad de opciones del campo select. Cantidad de etiquetas <OPTION>

Option
Hace referencia a cada una de sus opciones. Son por si mismas objetos.

options
Un array con cada una de las opciones del select.

selectedIndex
Es el índice de la opción que se encuentra seleccionada.

Aparte de las conocidas propiedades comunes a todos los elementos de formulario: form y name y type.

5.7.2.- Métodos del objeto select

Los métodos son solamente 2 y ya conocemos su significado.

blur()
Para retirar el foco de la aplicación de ese elemento de formulario.

focus()
Para poner el foco de la aplicación.

Objeto option

Tenemos que pararnos a ver también este objeto para entender bien el campo select. Recordamos que las option son las distintas opciones que tiene un select, expresadas con la etiqueta <OPTION>.

5.7.3.- Propiedades de option

Estos objetos sólo tienen propiedades, no tienen métodos. Vamos a verlas.

defaultSelected

Indica con un true o un false si esa opción es la opción por defecto. La opción por defecto se consigue con HTML colocando el atributo selected a un option.

index
El índice de esa opción dentro del select.

selected
Indica si esa opción se encuentra seleccionada o no.

text
Es el texto de la opción. Lo que puede ver el usuario en el select, que se escribe después de la etiqueta <OPTION>.

value
Indica el valor de la opción, que se introduce con el atributo VALUE de la etiqueta <OPTION>.

5.7.4.- Ejemplo de acceso a un select

Vamos a ver un ejemplo sobre cómo se accede a un select con Javascript, como podemos acceder a sus distintas propiedades y a la opción seleccionada.

Vamos a empezar viendo el formulario que tiene el select con el que vamos a trabajar. Es un select que sirve para valorar el web que estamos visitando.

```
<form name="fomul">
Valoración sobre este web:
<select name="miSelect">
<option value="10">Muy bien
<option value="5" selected>Regular
<option value="0">Muy mal
</select>
<br>
<br>
<input type=button value="Dime propiedades" onclick="dimePropiedades()">
</form>
```

Ahora vamos a ver una función que recoge las propiedades más significativas del campo select y las presenta en una caja alert.

```
function dimePropiedades(){
    var texto
    texto = "El numero de opciones del select: " + document.formul.miSelect.length
    var indice = document.formul.miSelect.selectedIndex
    texto += "nIndice de la opcion escogida: " + indice
    var valor = document.formul.miSelect.options[indice].value
    texto += "nValor de la opcion escogida: " + valor
    var textoEscogido = document.formul.miSelect.options[indice].text
    texto += "nTexto de la opcion escogida: " + textoEscogido
    alert(texto) }
```

Esta función crea una variable de texto donde va introduciendo cada una de las propiedades del select. La primera contiene el valor de la propiedad length del select, la segunda el índice de la opción seleccionada y las dos siguientes contienen el valor y el texto de la opción seleccionada. Para acceder a la opción seleccionada utilizamos el array options con el índice recogido en las segunda variable.

Podemos ver el ejemplo en funcionamiento aquí.

Podemos ver un ejemplo más practico sobre qué se puede hacer con un campo select en el reportaje de cómo hacer un navegador desplegable con Javascript.

5.7.5.- Propiedad value de un objeto select

Para acceder al valor seleccionado de un campo select podemos utilizar la propiedad value del campo select en lugar de acceder a partir del vector de options.

Para el anterior formulario sería algo parecido a esto.

```
formul.miSelect.value
```

Sin embargo, esta propiedad sólo está presente en navegadores Internet Explorer, por lo que es recomendable acceder utilizando el vector de options con el índice de la posición seleccionada si deseamos que la página sea compatible con todos los navegadores. Hemos querido añadir este punto para que, si alguna vez utilizamos o vemos utilizar este método de acceso, sepamos su pega y porque es mejor utilizar el vector de options.

Artículo por Miguel Angel Alvarez

5.8.- Control de elementos Textarea en Javascript

Los elementos textarea son los campos que permiten introducir varias líneas de texto. Aprendemos a controlarlos con programación Javascript.

Para acabar de describir todos los elementos de formularios vamos a ver el objeto textarea que es un elemento que presenta un lugar para escribir texto, igual que los campos text, pero con la particularidad que podemos escribir varias líneas a la vez.

El campo textarea se puede ver a continuación.

```
En los Textarea
Podemos colocar el texto
en varias líneas
```

Un campo textarea se consigue con la etiqueta <TEXTAREA>. Con el atributo name le podemos dar un nombre para acceder al campo textarea mediante Javascript. Otros atributos interesantes son cols y rows que sirven para indicar la anchura y altura del campo textarea en caracteres, cols indica el número de columnas y rows el de filas. aunque no se puede acceder a ellos con Javascript. El valor por defecto de un campo textarea se coloca entre las etiquteta <TEXTAREA> y su correspondiente cierre.

5.8.1.- Propiedades de textarea

Se puede ver una lista de las propiedades disponibles en un textarea a continuación, que son los mismos que un campo de texto.

defaultValue
Que contiene el valor por defecto del textarea.

value
Que contiene el texto que hay escrito en el textarea.

Además tiene las conocidas propiedades de elementos de formulario form, name y type.

5.8.2.- Métodos de textarea

Veamos una lista de los métodos, que son los mismos que en un campo de texto.

blur()
Para quitar el foco de la aplicación del textarea.

focus()
Para poner el foco de la aplicación en el textarea.

select()
Selecciona el texto del textarea.

Vamos a ver un ejemplo a continuación que presenta un formulario que tiene un textarea y un botón. Al apretar el botón se cuenta el número de caracteres y se coloca en un campo de texto.

Para acceder al número de caracteres lo hacemos a partir del la propiedad value del objeto textarea. Como value contiene un string podemos acceder a la propiedad length que tienen todos los strings para averiguar su longitud.

El código de la página se puede ver aquí.

```
<html>
<head>
    <title>Ejemplo textarea</title>
<script>
function cuenta(){
    numCaracteres = document.formul.textito.value.length
    document.formul.numCaracteres.value = numCaracteres
}
</script>
</head>
<body>
<form name="formul">
<textarea name=textito cols=40 rows=3>
Este es el texto por defecto
</textarea>
<br>
<br>
Número de caracteres <input type="Text" name="numCaracteres" size="4">
<br>
<br>
<input type=button value="Cuenta caracteres" onclick="cuenta()">
</form>
</body>
</html>
```

El ejemplo funcionando se puede ver en una página independiente.

Para quien desee profundizar, tenemos un artículo interesante que amplía el ejemplo anterior. Se trata de una cuenta de los caracteres del textarea a la vez que se está escribiendo dentro del campo. Se realiza gracias al tratamiento de eventos. Se puede ver el artículo en la dirección http://www.desarrolloweb.com/articulos/1348.php

Artículo por Miguel Angel Alvarez

Parte 6:
Eventos en Javascript

Los eventos son la base de la interactividad, de modo que en este apartado veremos cómo hacer páginas que respondan a las acciones del usuario. En Javascript podemos ejecutar código como respuesta a eventos, que son distintos tipos de acciones que el visitante puede realizar sobre la página o sobre sus elementos.

6.1.- Los eventos en Javascript

Una explicación sobre lo que son los eventos y como definir sus acciones asociadas en Javascript.

Los eventos son la manera que tenemos en Javascript de controlar las acciones de los visitantes y definir un comportamiento de la página cuando se produzcan. Cuando un usuario visita una página web e interactúa con ella se producen los eventos y con Javascript podemos definir qué queremos que ocurra cuando se produzcan.

Con javascript podemos definir qué es lo que pasa cuando se produce un evento como podría ser que un usuario pulse sobre un botón, edite un campo de texto o abandone la página.

El manejo de eventos es el caballo de batalla para hacer páginas interactivas, porque con ellos podemos responder a las acciones de los usuarios. Hasta ahora en este manual hemos podido ver muchos ejemplos de manejo de uno de los eventos de Javascript, el evento onclick, que se produce al pulsar un elemento de la página. Hasta ahora siempre hemos aplicado el evento a un botón, pero podríamos aplicarlo a otros elementos de la página.

6.1.1.- Cómo se define un evento

Para definir las acciones que queremos realizar al producirse un evento utilizamos los manejadores de eventos. Existen muchos tipos de manejadores de eventos, para muchos tipos de acciones del usuario. El maneador de eventos se coloca en la etiqueta HTML del elemento de la página que queremos que responda a las acciones del usuario.

Por ejemplo tenemos el manejador de eventos onclick, que sirve para describir acciones que queremos que se ejecuten cuando se hace un click. Si queremos que al hacer click sobre un botón pase alguna cosa, escribimos el manejador onclick en la etiqueta <INPUT type=button> de ese botón. Algo parecido a esto.

```
<INPUT type=button value="pulsame" onclick="sentencias_javascript...">
```

Se coloca un atributo nuevo en la etiqueta que tiene el mismo nombre que el evento, en este caso onclick. El atributo se iguala a las sentencias Javascript que queremos que se ejecuten al producirse el evento.

Cada elemento de la página tiene su propia lista de eventos soportados, vamos a ver otro ejemplo de manejo de eventos,

esta vez sobre un menú desplegable, en el que definimos un comportamiento cuando cambiamos el valor seleccionado.

```
<SELECT onchange="window.alert('Cambiaste la selección')">
<OPTION value="opcion1">Opcion 1
<OPTION value="opcion2">Opcion 2
</SELECT>
```

En este ejemplo cada vez que se cambia la opción muestra una caja de alerta. Podemos verlo en una página aparte.

Dentro de los manejadores de eventos podemos colocar tantas instrucciones como deseemos, pero siempre separadas por punto y coma. Lo habitual es colocar una sola instrucción, y si se desean colocar más de una se suele crear una función con todas las instrucciones y dentro del manejador se coloca una sola instrucción que es la llamada a la función.

Vamos a ver cómo se colocarían en un manejador varias instrucciones.

```
<input type=button value=Pulsame
   onclick="x=30; window.alert(x); window.document.bgColor = 'red'">
```

Son instrucciones muy simples como asignar a x el valor 30, hacer una ventana de alerta con el valor de x y cambiar el color del fondo a rojo. Podemos ver el ejemplo en una página aparte.

Sin embargo, tantas instrucciones puestas en un manejador quedan un poco confusas, habría sido mejor crear una función así.

```
<script>
function ejecutaEventoOnclick(){
    x = 30
    window.alert(x)
    window.document.bgColor = 'red'
}
</script>
```

```
<FORM>
<input type=button value=Pulsame onclick="ejecutaEventoOnclick()">
</FORM>
```

Ahora utilizamos más texto para hacer lo mismo, pero seguro que a la mayoría les parece más claro este segundo ejemplo. Si se desea, se puede ver esta última página en una ventana aparte

6.1.2.- Jerarquía desde el objeto window

En los manejadores de eventos se tiene que especificar la jerarquía entera de objetos del navegador, empezando siempre por el objeto window. Esto es necesario porque hay algún browser antiguo que no sobreentiende el objeto window cuando se escriben sentencias Javascript vinculadas a manejadores de eventos.

Artículo por Miguel Angel Alvarez

6.2.- Los manejadores de eventos en Javascript

Lista de los manejadores de eventos más habituales del lenguaje Javascript, junto con una descripción de cada uno.

Ahora vamos a ver una lista de los **manejadores de eventos que hay disponibles en Javascript**, ofreciendo una pequeña descripción de cada uno.

Si queremos saber previamente qué es un evento y como se tratan en Javascript, podemos consultar el artículo anterior del manual: Los eventos en Javascript

Nota: Estos manejadores de eventos son los más comunes, presentes en Javascript 1.2 de Netscape Navigator. Existen otros manejadores que también son muy interesantes y veremos más adelante en capítulos de temas avanzados de eventos.

La lista de manejadores de eventos contiene el nombre del manejador en negrita, su descripción y finalmente la versión de Javascript que incorporó dicho manejador.

onabort
Este evento se produce cuando un usuario detiene la carga de una imagen, ya sea porque detiene la carga de la página o porque realiza una acción que la detiene, como por ejemplo irse de la página.
Javascript 1.1

onblur
Se desata un evento onblur cuando un elemento pierde el foco de la aplicación. El foco de la aplicación es el lugar donde está situado el cursor, por ejemplo puede estar situado sobre un campo de texto, una página, un botón o cualquier otro elemento.
Javascript 1.0

onchange
Se desata este evento cuando cambia el estado de un elemento de formulario, en ocasiones no se produce hasta que el usuario retira el foco de la aplicación del elemento. Javascript 1.0
Javascript 1.0

onclick
Se produce cuando se da una pulsación o clic al botón del ratón sobre un elemento de la página, generalmente un botón o un enlace.
Javascript 1.0

ondragdrop
Se produce cuando un usuario suelta algo que había arrastrado sobre la página web.
Javascript 1.2

onerror
Se produce cuando no se puede cargar un documento o una imagen y esta queda rota.
Javascript 1.1

onfocus
El evento onfocus es lo contrario de onblur. Se produce cuando un elemento de la página o la ventana ganan el foco de la aplicación.
Javascript 1.0

onkeydown
Este evento se produce en el instante que un usuario presiona una tecla, independientemente que la suelte o no. Se produce en el momento de la pulsación.
Javascript 1.2

onkeypress
Ocurre un evento onkeypress cuando el usuario deja pulsada una tecla un tiempo determinado. Antes de este evento se produce un onkeydown en el momento que se pulsa la tecla..
Javascript 1.2

onkeyup
Se produce cuando el usuario deja de apretar una tecla. Se produce en el momento que se libera la tecla.
Javascript 1.2

onload
Este evento se desata cuando la página, o en Javascript 1.1 las imágenes, ha terminado de cargarse.
Javascript 1.0

onmousedown
Se produce el evento onmousedown cuando el uuario pulsa sobre un elemento de la página. onmousedown se produce en el momento de pulsar el botón, se suelte o no.
Javascript 1.2

onmousemove
Se produce cuando el ratón se mueve por la página.
Javascript 1.2

onmouseout
Se desata un evento onmuoseout cuando el puntero del ratón sale del área ocupada por un elemento de la página.
Javascript 1.1

onmouseover
Este evento se desata cuando el puntero del ratón entra en el área ocupada por un eolemento de la página.
Javascript 1.0

onmouseup
Este evento se produce en el momento que el usuario suelta el botón del ratón, que previamente había pulsado.
Javascript 1.2

onmove
Evento que se ejecuta cuando se mueve la ventana del navegador, o un frame.
Javascript 1.2

onresize
Evento que se produce cuando se redimensiona la ventana del navegador, o el frame, en caso de que la página los tenga.

Javascript 1.2

onreset
Este evento está asociado a los formularios y se desata en el momento que un usuario hace clic en el botón de reset de un formulario.
Javascript 1.1

onselect
Se ejecuta cuando un usuario realiza una selección de un elemento de un formulario.
Javascript 1.0

onsubmit
Ocurre cuando el visitante apreta sobre el botón de enviar el formulario. Se ejecuta antes del envío propiamente dicho.
Javascript 1.0

onunload
Al abandonar una página, ya sea porque se pulse sobre un enlace que nos lleve a otra página o porque se cierre la ventana del navegador, se ejecuta el evento onunload.
Javascript 1.0

Artículo por Miguel Angel Alvarez

6.3.- Ejemplos de eventos en Javascript. Onabort

Vemos enlaces a diversas aplicaciones prácticas donde se tratan eventos y ofrecemos un nuevo ejemplo para el evento onabort.

A lo largo de los manuales I y II de Javascript, así como del Taller, hemos mostrado muchos ejemplos de utilización de los manejadotes de eventos. Aquí veremos ejemplos sencillos que se nos ocurren para utilizar otros manejadotes que no hemos visto todavía, aunque antes podemos hacer una lista de algunos ejemplos publicados anteriormente que deberían servir de ayuda para ir captando la práctica de el manejo de eventos.

- Acceso por clave con Javascript (Evento onclick)
- Rollover con Javascript (Eventos onmuoseover y onmouseout)
- Navegador desplegable (Evento onchange)
- Calculadora sencilla (Evento onclick)
- Confirmación del envío de formulario (Evento onclick)
- Posicionarse en un select (Evento onkeypress)
- Inhibir campo de formulario (Evento onfocus)
- Cuenta caracteres de un textarea (Eventos onkeydown y onkeyup)

6.3.1.- Evento onabort

Veamos un primer ejemplo, en este caso sobre el evento onabort. Este evento se activa al cancelarse la carga de una página, ya sea porque se pulsa el botón de cancelar o porque el usuario se marcha de la página por otro enlace.

Este ejemplo contiene una imagen que tiene el evento onabort asignado para que se ejecute una función en caso de que la imagen no llegue a cargarse. La función informa al usuario de que la imagen no se ha llegado a cargar y le pregunta si desea cargarla otra vez. Si el usuario contesta que sí, entonces se pone a descargar la imagen otra vez. Si dice que no, no hace nada. La pregunta se hace con una caja confirm de Javascript.

```
<html> <head>
    <title>Evento onabort</title>

<script>
function preguntarSeguir(){
   respuesta = confirm ("Has detenido la carga de la página y hay una imagen que no estás viendo.n¿Deseas cargar la imagen?")
   if (respuesta)
      document.img1.src = "http://ipaginate.iespana.es/ipaginate/desarrollogrande.gif"
}
</script>

</head>
<body>
<img name=img1 src="http://ipaginate.iespana.es/ipaginate/desarrollogrande.gif" width=500 height=458 alt="Imagen que pesa 115K" border="0" onabort="preguntarSeguir()">
<br>
Pulsa el botón de parar la carga de la página y se pondrá en marcha el evento onerror

</body>
</html>
```

Este ejemplo estaría bien si siempre se detuviese la carga por pulsar el botón de cancelar, pero si lo que pasa es que el usuario ha cancelado por irse a otra página a través de un enlace, saldrá la caja de confirmación pero no ocurrirá nada independientemente de lo que se responda y el navegante se marchará irremediablemente a la nueva página.

Se puede ver en una página aparte.

Artículo por Miguel Angel Alvarez

6.4.- Ejemplo del evento onblur en Javascript

Script en Javascript que muestra el trabajo con el evento onblur. Se comprueba la validez de un dato al salir del campo de texto donde está escrito.

Onblur se activa cuando el usuario retira el foco de la aplicación de un elemento de la página. El foco de la aplicación es el lugar donde está el cursor.

Si por ejemplo, estamos situados en un campo de texto y nos vamos de dicho campo, ya sea porque pulsamos con el ratón en otro campo del formulario o en un área vacía, ya sea porque el usuario a apretado el botón tabulador (Tab) que mueve el foco hasta el siguiente elemento de la página.

Si yo deseo que, al situarse fuera de un campo de texto, se compruebe que el valor introducido es correcto puedo utilizar onblur y llamar a una función que compruebe si el dato es correcto. Si no es correcto puedo obligar al foco de la aplicación a situarse nuevamente sobre el campo de texto y avisar al usuario para que cambie dicho valor.

Puede ser una manera interesante de asegurarnos que en un campo de texto se encuentra un valor válido. Aunque tiene alguna pega, como veremos más adelante.

Vamos a empezar por un caso sencillo, en el que solamente deseamos comprobar un campo de texto para asegurarnos que es un número entero.

> **Referencia:** La función que valida un entero la hemos explicado en un taller anterior de Javascript: Validar entero en campo de formulario.

```
<html>
<head>
    <title>Evento onblur</title>
```

```
<script>
function validarEntero(valor){
   //intento convertir a entero.
   //si era un entero no le afecta, si no lo era lo intenta convertir
   valor = parseInt(valor)

   //Compruebo si es un valor numérico
   if (isNaN(valor)) {
      //entonces (no es numero) devuelvo el valor cadena vacia
      return ""
   }else{
      //En caso contrario (Si era un número) devuelvo el valor
      return valor
   }
}

function compruebaValidoEntero(){
   enteroValidado = validarEntero(document.f1.numero.value)
   if (enteroValidado == ""){
      //si era la cadena vacía es que no era válido. Lo aviso
      alert ("Debe escribir un entero!")
      //selecciono el texto
      document.f1.numero.select()
      //coloco otra vez el foco
      document.f1.numero.focus()
   }else
      document.f1.numero.value = enteroValidado
```

```
}
</script>
</head>
<body>
<form name=f1>
Escriba un número entero: <input type=text name=numero size=8 value="" onblur="compruebaValidoEntero()">
</form>

</body>
</html>
```

Al salirse del campo de texto (onblur) se ejecuta compruebaValidoEntero(), que utiliza la función validarEntero, explicada en un taller de Javascript. Si el valor devuelto por la función no es el de un entero, en este caso se recibiría una cadena vacía, muestra un mensaje en una caja alert, selecciona el texto escrito en la caja y coloca el foco de la aplicación en la caja de texto, para que el usuario coloque otro valor.

Hasta que el visitante no escriba un número entero en el campo de texto el foco de la aplicación permanecerá en el campo y continuará recibiendo mensajes de error.

Podemos ver este ejemplo en marcha en una página aparte.

Artículo por Miguel Angel Alvarez

6.5.- Continuación del ejemplo de onblur, para validar varios campos de texto.

Hacemos un ejemplo de validación de campos de un formulario utilizando como base el evento onblur y solucionando un problema de bucle infinito.

Hemos visto en el ejemplo del método onblur relatado anteriormente una posible técnica para comprobar los datos de un campo de formulario. Ahora vamos a ver cómo evolucionar esta técnica si tenemos más de un campo a validar, dado que se puede complicar bastante el problema.

De hecho, antes de leer nuestra solución propuesta, creo que sería un buen ejercicio a realizar por el lector la práctica de hacer ese mismo ejemplo para dos campos y trabajar un poco con la página a ver si encontramos algún problema.

Muy probablemente nos encontraremos con un curioso bucle infinito que nos va a dar más de un quebradero de cabeza para solucionarlo.

En la práctica, el lector puede intentar validar un número entero y un código postal. Para validar un código postal necesitamos comprobar que es una cadena de texto compuesta por 5 caracteres y todos son enteros, por lo menos para los códigos en España.

Por si alguien lo quiere intentar, la función para validar un código postal sería algo parecido a esto:

```
function ValidoCP(){
    CPValido=true
    //si no tiene 5 caracteres no es válido
   if (document.f1.codigo.value.length != 5)
         CPValido=false
      else{
         for (i=0;i<5;i++){
            CActual = document.f1.codigo.value.charAt(i)
            if (validarEntero(CActual)==""){
               CPValido=false
               break;
            }
```

```
            }
        }
    return CPValido
}
```

Simplemente se encarga de comprobar que el campo de texto contiene 5 caracteres y hacer un recorrido por cada uno de los caracteres para comprobar que todos son enteros. Al principio se supone que el código postal es correcto, colocando la variable CPValido a true, y si alguna comprobación falla se cambia el estado correcto a incorrecto, pasando dicha variable a false.

Se puede probar a montar el ejemplo con dos campos... nosotros ahora vamos a dar una solución al problema bastante complicadilla, ya que incluimos instrucciones para evitar el efecto del bucle infinito. No vamos a ver el ejemplo que daría el error, lo dejamos para el que desee intentarlo por si mismo y recomendamos hacerlo porque así comprenderemos mejor el siguiente código.

```
<html>
<head>
    <title>Evento onblur</title>

<script>
avisado=false
function validarEntero(valor){
    //intento convertir a entero.
    //si era un entero no le afecta, si no lo era lo intenta convertir
     valor = parseInt(valor)

    //Compruebo si es un valor numérico
    if (isNaN(valor)) {
      //entonces (no es numero) devuelvo el valor cadena vacia
      return ""
    }else{
      //En caso contrario (Si era un número) devuelvo el valor
      return valor
    }
}

function compruebaValidoEntero(){
  enteroValidado = validarEntero(document.f1.numero.value)
  if (enteroValidado == ""){
    //si era la cadena vacía es que no era válido. Lo aviso
    if (!avisado){
      alert ("Debe escribir un entero!")
      //selecciono el texto
      document.f1.numero.select()
      //coloco otra vez el foco
      document.f1.numero.focus()
      avisado=true
      setTimeout('avisado=false',50)
    }
  }else
    document.f1.numero.value = enteroValidado
}

function compruebaValidoCP(){
  CPValido=true
  //si no tiene 5 caracteres no es válido
  if (document.f1.codigo.value.length != 5)
    CPValido=false
```

```
  else{
    for (i=0;i<5;i++){
      CActual = document.f1.codigo.value.charAt(i)
      if (validarEntero(CActual)==""){
        CPValido=false
        break;
      }
    }
  }
  if (!CPValido){
    if (!avisado){
      //si no es valido, Lo aviso
      alert ("Debe escribir un código postal válido")
      //selecciono el texto
      document.f1.codigo.select()
      //coloco otra vez el foco
      //document.f1.codigo.focus()
      avisado=true
      setTimeout('avisado=false',50)
    }
  }
}
</script>

</head>
<body>

<form name=f1>
Escriba un número entero: <input type=text name=numero size=8 value="" onblur="compruebaValidoEntero()">
<br>
Escriba un código postal: <input type=text name=codigo size=8 value="" onblur="compruebaValidoCP()"> *espera una cadena con 5 carácteres numéricos

</form>

</body>
</html>
```

Este ejemplo sigue la guía del primer ejemplo de onblur de este artículo, incluyendo un nuevo campo a validar.

Para solucionar el tema del bucle infinito, que habréis podido investigar por vosotros mismos y en el que se mostraban una caja de alerta tras otra indefinidamente, hemos utilizado una variable llamada avisado que contiene un true si ya se ha avisado de que el campo estaba mal y un false si no se ha avisado todavía.

Cuando se va a mostrar la caja de alerta se comprueba si se ha avisado o no al usuario. Si ya se avisó no se hace nada, evitando que se muestren más cajas de alerta. Si no se había avisado todavía se muestra la caja de alerta y se coloca el foco en el campo que era incorrecto.

Para restituir la variable avisado a false, de modo que la próxima vez que se escriba mal el valor se muestre el mensaje correspondiente, se utiliza el método setTimeout, que ejecuta la instrucción con un retardo, en este caso de 50 milisegundos. Lo suficiente para que no se meta en un bucle infinito.

> **Nota:** Después de todos los parches que hemos tenido que colocar para que este evento se comporte correctamente para cumplir el cometido deseado, es posible que no merezca la pena utilizarlo para este cometido. Podemos hacer uso del evento onchange, o comprobar todos los campos de una sola vez cuando el usuario ha decidido enviarlo.

Podemos ver en marcha este ejemplo en una página aparte.

Artículo por Miguel Angel Alvarez

6.6.- Elementos de formulario select asociados

Cómo hacer con Javascript que un elemento de formulario select cambie sus opciones cuando cambie otro elemento select de la página. De modo que cada opción de un select tenga un grupo de opciones posibles para el otro select.

Vamos a conocer uno de los trucos más solicitados de Javascript, que tiene mucha relación con el tema de formularios y donde se utiliza el evento onchange de Javascript. Es un ejemplo sobre cómo realizar una página con un par de selects donde, según el valor escogido en uno de ellos, cambien las opciones posibles del otro select.

Lo mejor para ver lo que vamos a hacer es ver una página web donde se muestra en funcionamiento el script. Para ver su funcionamiento debemos cambiar la selección del primer select y comprobaremos como las opciones del segundo select cambian automáticamente.

El ejemplo que hemos ilustrado utiliza provincias y países. Al escoger en el primer select un país, en el segundo debe mostrarnos las provincias de ese país para que escojamos una provincia, pero sólo una que tenga que esté en el país seleccionado en primer término.

6.6.1.- Conocer el objeto select y los option

Es importante conocer los objetos de formulario select y los option. Los select corresponden con las cajas de selección desplegables y los option con cada una de las opciones de la caja desplegable. Podemos ver un artículo que habla de ello.

En concreto nos interesa hacer varias cosas que tienen que ver con extraer el valor de un select en cualquier momento, fijar su número de opciones y, para cada opción, colocar su valor y su texto asociado. Todo esto aprenderemos a hacerlo en este ejemplo.

> **Referencia:** Para conocer el trabajo con formularios y la jerarquía de objetos javascript (Todo eso es la base del trabajo con los elementos de las páginas en Javascript) debemos haber leer el manual de Javascript II.

6.6.2.- Modo de llevar a cabo el problema

Para empezar, vamos a utilizar un formulario con dos selects, uno para el país y otro para la provincia.

```
<form name="f1">
<select name=pais onchange="cambia_provincia()">
<option value="0" selected>Seleccione...
<option value="1">España
<option value="2">Argentina
<option value="3">Colombia
<option value="4">Francia
</select>
```

<select name=provincia>
<option value="-">-
</select>
</form>

Nos fijamos en el select asociado al país de este formulario que, cuando se cambia la opción de país, se debe llamar a la función cambia_provincia(). Veremos más adelante esta función, ahora es importante fijarse que está asociada al evento

onchange que se activa cuando cambia la opción en el select.

Todo lo demás será código Javascript. Empezamos definiendo un montón de arrays con las provincias de cada país. En este caso tenemos sólo 4 países, entonces necesitaré 4 arrays. En cada array tengo la lista de provincias de cada país, colocada en cada uno de los elementos del array. Además, dejaré una primera casilla con un valor "-" que indica que no se ha seleccionado ninguna provincia.

```
var provincias_1=new Array("-","Andalucía","Asturias","Baleares","Canarias","Castilla y
León","Castilla-La Mancha","...")
var provincias_2=new Array("-","Salta","San Juan","San Luis","La Rioja","La Pampa","...")
var provincias_3=new Array("-","Cali","Santamarta","Medellin","Cartagena","...")
var provincias_4=new Array("-","Aisne","Creuse","Dordogne","Essonne","Gironde ","...")
```

Hay que fijarse que los índices del array de cada país se corresponden con los del select del país. Por ejemplo, la opción España, tiene el valor asociado 1 y el array con las provincias de España se llama provincias_1.

El script se completa con una función que realiza la carga de las provincias en el segundo select. El mecanismo realiza básicamente estas acciones:

- Detecto el país que se ha seleccionado
- Si el valor del país no es 0 (el valor 0 es cuando no se ha seleccionado país)
    - Tomo el array de provincias adecuado, utilizando el índice del país.
    - Marco el número de opciones que debe tener el select de provincias
    - Para cada opcion del select, coloco su valor y texto asociado, que se hace corresponder con lo indicado en el array de provincias.
- SI NO (El valor de país es 0, no se ha seleccionado país)
    - Coloco en el select de provincia un único option con el valor "-", que significaba que no había provincia.
- Coloco la opción primera del select de provincia como la seleccionada.

La función tiene el siguiente código. Está comentado para que se pueda entender mejor.

```
function cambia_provincia(){
    //tomo el valor del select del pais elegido
    var pais
    pais = document.f1.pais[document.f1.pais.selectedIndex].value
    //miro a ver si el pais está definido
    if (pais != 0) {
       //si estaba definido, entonces coloco las opciones de la provincia correspondiente.
       //selecciono el array de provincia adecuado
       mis_provincias=eval("provincias_" + pais)
       //calculo el numero de provincias
       num_provincias = mis_provincias.length
       //marco el número de provincias en el select
       document.f1.provincia.length = num_provincias
       //para cada provincia del array, la introduzco en el select
       for(i=0;i<num_provincias;i++){
          document.f1.provincia.options[i].value=mis_provincias[i]
          document.f1.provincia.options[i].text=mis_provincias[i]
       }
    }else{
       //si no había provincia seleccionada, elimino las provincias del select
       document.f1.provincia.length = 1
       //coloco un guión en la única opción que he dejado
       document.f1.provincia.options[0].value = "-"
       document.f1.provincia.options[0].text = "-"
    }
    //marco como seleccionada la opción primera de provincia
    document.f1.provincia.options[0].selected = true
}
```

Podemos ver una página con el ejemplo en funcionamiento.

Artículo por Miguel Angel Alvarez

6.7.- Evento onunload de Javascript

Ejemplo de uso del evento onunload en Javascript para abrir una ventana secundaria cuando el usuario abandone la página.

Veamos un ejemplo del evento onunload, que, recordamos, se activa cuando el usuario ha abandona la página web. Por tanto, onunload sirve para ejecutar una acción cuando el usuario se marcha de la página, ya sea porque pulsa un enlace que le lleva fuera de la página o porque cierra la ventana del navegador.

El ejemplo que deseamos mostrar sirve para abrir una página web en otra ventana cuando el usuario abandona la página. De este modo actúan muchos de los molestos popups de las páginas web, abriéndose justo cuando abandonamos el sitio que estábamos visitando.

```
<html>
<head>
    <title>Abre al salir</title>
    <script>
    function abreventana(){
       window.open("http://www.google.es","venta","")
    }
    </script>
</head>
```

<body onunload="abreventana()">

DW!!
</body>
</html>

El ejemplo es tan sencillo que casi sobran las explicaciones. Simplemente creamos una función que abre una ventana secundaria y la asociamos con el evento onunload, que se coloca en la etiqueta <body>.

Se puede ver en marcha en una página aparte.

Referencia: Si no tenemos una base de Javascript nos vendrá muy bien acceder a nuestra sección Javascript a fondo.

Si deseamos conocer más cosas de los eventos.
Si deseamos saber más sobre abrir ventanas.

Artículo por Miguel Angel Alvarez

6.8.- Evento onload de Javascript

Con el evento onload podemos ejecutar acciones justo cuando se han terminado de cargar todos los elementos de la página.

El evento onload de Javascript se activa cuando se termina de cargar la página. Se ha de colocar en la etiqueta <body>, aunque en versiones modernas de Javascript, también lo aceptan otros elementos como las imágenes.

Con el evento onload podemos ejecutar acciones justo cuando se han terminado de cargar todos los elementos de la página. Es un evento bastante utilizado pues es muy habitual que se deseen llevar a cabo acciones en ese preciso

instante. En nuestro ejemplo vamos a ver cómo podríamos hacer un script para motivar a nuestros visitantes a que nos voten en un ranking cualquiera de páginas web.

La idea es que la página se cargue entera y, una vez está cargada, aparezca una ventana de Javascript donde se proponga al visitante votar a la página. Es interesante esperar a que termine de cargar la página entera para que el visitante pueda ver la web que se propone votar, antes de que realmente le pidamos su voto.

El código sería el siguiente:

```
<html>
<head>
    <title>Votame!!</title>
<script language="JavaScript">
function pedirVoto(){
    if (confirm("Esta página está genial (ya la puedes ver). ¿Me das tu voto?")){
       window.open("http://www.loquesea.com/votar.php?idvoto=12111","","")
    }
}
</script>
</head>

<body onload="pedirVoto()">
<h1>Página SuperChula</h1>
<br>
Esta página está muy bonita. Soy su autor y te puedo asegurar que no hay muchas páginas con tanta calidad en Internet
<br>
<br>
<a href="#">Entrar</a>

</body>
</html>
```

Nos fijamos que en la etiqueta <body> tenemos el evento onload="pedirVoto()". Es decir, que cuando se cargue la página se llamará a una función llamada pedirVoto(), que hemos definido en el bloque de script que tenemos en la cabecera.

La función pedirVoto() utiliza una caja confirm para saber si realmente el usuario desea votarnos. La función confirm() muestra una caja con un texto y dos botones, para aceptar o cancelar. El texto es lo que se recibe por parámetro. Dependiendo de lo que se pulse en los botones, la función confirm() devolverá un true, si se apretó el botón aceptar, o un false, en caso de que se pulsase sobre cancelar.

La función confirm() está a su vez metida dentro de un bloque condicional if, de modo que, dependiendo de lo que se pulsó en la caja confirm, el if se evaluará como positivo o negativo. En este caso sólo se realizan acciones si se pulsó sobre aceptar.

Para acceder a la página donde se contabiliza nuestro voto tenemos el método window.open(), que sirve para abrir ventanas secundarias o popups. Mostramos la página donde se vota en una ventana secundaria para que el visitante no se marche de nuestra web, ya que acaba de entrar y no deseamos que se vaya ya.

Con esto queda más o menos ilustrado cómo hacer uso del evento onload. Seguro que en vuestras creaciones habrá muchos más casos donde utilizarlo.

Artículo por Miguel Angel Alvarez

6.9.- Manejadores de eventos en Javascript 1.3

Listado de todos los manejadores de eventos en el lenguaje Javascript 1.3.

En artículos anteriores de DesarrolloWeb.com vimos diversas informaciones relativas a los eventos en Javascript. En esta ocasión vamos a proporcionar un listado que sirva como referencia para los programadores con Javascript 1.3.

En concreto, vimos con anterioridad estos artículos que conviene repasar:

- Los eventos en Javascript
- Manejadores de eventos de Javascript

En el artículo de manejadores de eventos de Javascript vimos los eventos de las versiones de Javascript 1.0, 1.1 y 1.2, pero en este caso vamos a presentar el listado de los manejadores de eventos de Javascript 1.3.

Cada evento tiene un nombre, por ejemplo "click". Los manejadores eventos, que son usados para invocar una serie de comandos cuando se produce un evento, tienen siempre la palabra "on" seguida del nombre del evento. Por ejemplo, "onclick".

6.9.1.- Manejadores de eventos (Event Handlers) en Javascript 1.3

Abort (onabort): Se lanza cuando se abortó la carga de un elemento de la página, por ejemplo una imagen.

Blur (onblur): Se procesa este evento cuando un elemento de la página, generalmente un elemento de formulario, pierde el foco de la aplicación.

Change (onchange): Este evento se produce cuando el usuario cambia el contenido de un elemento de formulario, tipo select, input o textarea.

Click (onclick): Se lanza cuando el usuario hace clic sobre un elemento de la página, que puede ser un botón, un enlace, etc.

DblClick (ondblclick): Este evento es lanzado cuando el usuario hace doble click en un elemento de formulario o un link.

DragDrop (ondragdrop): Este evento se produce cuando el usuario arrastra y suelta un objeto en la ventana del navegador.

Error (onerror): producido cuando hubo un error en la carga de un elemento de la página o de un documento.

Focus (onfocus): Se produce este evento cuando un elemento de la página, generalmente un elemento de formulario, gana el foco de la aplicación.

KeyDown (onkeydown): Este evento se lanza cuando el usuario pulsa una tecla.

KeyPress (onkeypress): Se lanza el evento onkeypress cuando el usuario pulsa o mantiene pulsada una tecla.

KeyUp (onkeyup): Se produce cuando el usuario suelta una tecla que tenía pulsada.

Load (onload): Se ejecuta cuando se termina de cargar la página, o bien todos los frames del conjunto de FRAMESET.

MouseDown (onmousedown): Este evento se produce cuando el usuario aprieta el botón del ratón.

MouseMove (onmousemove): Se ejecuta cuando el usuario mueve el ratón.

MouseOut (onmouseout): Se lanza cuando el usuario retira el puntero del ratón. Por ejemplo, si colocamos onmouseout sobre una imagen, el evento se lanzaría en el momento que el usuario sale con el puntero del ratón de esa imagen.

MouseOver (onmouseover): Este evento se desata cuando el usuario mueve el puntero del ratón sobre un elemento. Imaginemos que colocamos este evento en una imagen, entonces se lanza, una sola vez, en el momento de entrar con el puntero en el área que ocupa esa imagen.

MouseUp (onmouseup): Este evento se produce cuando el usuario suelta o deja de apretar el botón del ratón.

Move (onmove): Se produce cuando el usuario o un script mueven la ventana del navegador.

Reset (onreset): El evento se ejecuta cuando se resetea el contenido de un formulario, haciendo clic en un botón de reset del formulario, si es que lo tiene.

Resize (onresize): Se lanza cuando el usuario, o un script, alteran el tamaño de una ventana del navegador o de un frame.

Select (onselect): El evento se produce cuando el usuario selecciona un texto de un textarea o bien de un campo de texto normal.

Submit (onsubmit): Se lanza cuando se aprieta el botón de submitir de un formulario, así que permite ejecutar código cuando el usuario envía el formulario.

UnLoad (onunload): Evento que se produce cuando el usuario sale de un documento, osea, al salir de la página web, ya sea por pulsar un enlace que lleva a otra página o por cerrar la ventana del navegador.

Artículo por Miguel Angel Alvarez

Parte 7:
Epílogos a la segunda parte del Manual de Javascript

Diversos artículos con informaciones interesantes para completar la formación y para saber por dónde se puede continuar aprendiendo.

7.1.- Cláusulas try .catch: detectar y cazar errores en Javascript

Vemos aspectos básicos sobre la utilización de las cláusulas try y catch para detectar y cazar errores javascript, sin que los errores sean tratados por defecto.

Muchos lenguajes de programación utilizan las cláusulas try .catch para cazar errores y realizar cosas cuando ocurran, por ello, lo que vamos a comentar aquí para Javascript puede resultar muy familiar a los programadores. Estas cláusulas las podemos utilizar para tratar de ejecutar una porción de código, que sabemos que podría desatar un error en tiempo de ejecución.

Cuando ocurre un error en Javascript, se hace un tratamiento determinado (mostrar el error al usuario, ya sea mediante un mensaje o la consola Javascript, o simplemente mostrar un icono y detener la ejecución de ese código). Nosotros con try .catch podemos evitar que el error se trate de manera predeterminada y que se realicen unas acciones determinadas ante el error. Además, podemos conseguir que el código se siga ejecutando sin ningún problema.

Con try especificamos una serie de sentencias Javascript que vamos a tratar de ejecutar. Con catch especificamos lo que queremos realizar si es que se ha cazado un error en el bloque try. La sintaxis de utilización es la siguiente:

```
try {
   //intento algo que puede producir un error
}catch(mierror){
   //hago algo cuando el error se ha detectado
}
```

El Bloque try se ejecuta tal cual, hasta que un posible error se ha detectado.

- Si no se detecta un error durante la ejecución del bloque try, el catch se deja del lado y no se realiza.
- En caso que sí se detecte un error en el bloque try, se ejecuta las sentencias que teníamos en el catch.

Si nos fijamos, podemos ver que el bloque catch recibe un dato, que en este caso hemos almacenado en la variable "mierror". Esa variable contiene información sobre el error detectado, que puede ser útil para realizar el tratamiento al error. Veamos este código:

```
try {
   //intento algo que puede producir un error
   funcion_que_no_existe()
}catch(mierror){
   alert("Error detectado: " + mierror.description)
}
```

Cuando se ejecute el try, se detectará un error, al tratar de ejecutar una función que no existe. Entonces se ejecutará la cláusula catch, mostrando el error que se ha detectado. Si nos fijamos, se muestra una descripción del error detectado mediante mierror.description.

Pero la propiedad description del error sólo existe en Internet Explorer. En Firefox, para mostrar una descripción del error lo hacemos directamente con la variable. Así:

```
try {
   //intento algo que puede producir un error
   funcion_que_no_existe()
}catch(mierror){
   alert("Error detectado: " + mierror)
}
```

Entonces, para hacer un bloque try .catch como este que funcione en los dos navegadores deberíamos hacer esto:

```
try {
   //intento algo que puede producir un error
   funcion_que_no_existe()
}catch(mierror){
   if (mierror.description){
      alert("Error detectado: " + mierror.description)
   }else{
      alert("Error detectado: " + mierror)
   }
}
```

7.1.1.- Lanzar una excepción nosotros mismos

También, dentro de un bloque try, podemos lanzar nosotros mismos una excepción, sin que tenga por qué haberse producido un error. Esto lo hacemos con la sentencia trow.

```
try {
throw "miexcepcion"; //lanza una excepción
}
catch (e) {
//tratamos la excepción
alert(e);
}
```

Este código, válido tanto para Internet Explorer como Firefox, lanza una excepción en el bloque try. Luego, en el bloque catch, se muestra la excepción que se ha detectado.

Artículo por Miguel Angel Alvarez

7.2.- Hojas de referencia Javascript

Varias chuletas u hojas de referencia de Javascript, Con un listado de métodos funciones y usos de Javascript para rápida consulta.

En DesarrolloWeb.com ofrecemos diversos manuales para aprender Javascript, que se pueden consultar todos a través de la sección Javascript a fondo. En esta sección encontraréis numerosos textos explicativos para aprender a trabajar con el lenguaje, tanto para dar los primeros pasos como realizar operaciones más complicadas.

Nuestros manuales vienen con explicaciones detalladas sobre Javascript, pero siempre es bueno tener a mano unas "chuletas", o más dicho de forma más elegante, hojas de referencia rápida. En éstas se encuentran multitud de nombres de funciones, métodos y distintos usos del lenguaje, que son repetidos a menudo durante la programación.

En Internet se pueden encontrar diversas hojas de referencia muy útiles, completas y bien presentadas, que nos pueden ayudar en nuestro día a día con el lenguaje, si las tenemos impresas en papel y a mano para consulta. Lo bueno es que las ofrecen programadores de manera gratuita, con lo que podemos verlas y si nos parece que pueden ayudarnos, se pueden imprimir libremente. Os paso algunos enlaces interesantes para encontrar estas chuletas de Javascript:

Javascript Quick Reference Card
Chuleta de referencia Javascript escrita en dos páginas, con conceptos bastante básicos sobre estructuras de control, operadores, etc, así como objetos y métodos.

JavaScript and Browser Objects Quick Reference
8 páginas con un listado de todos los objetos del navegador y sus propiedades y métodos. Está sacada del libro "Javascript Bible".

Javascript Cheat Sheet:
En una sola página se muestran algunas notas sobre sintaxis, los métodos básicos de Javascript, eventos, los métodos DOM e incluso algunas ayudas sobre Ajax.

JavaScript in one page
No se podría imprimir en una sola página, pero muestra una completa tabla sobre Javascript, de la que se podría incluso aprender con las escuetas explicaciones.

Referencia Javascript
Otra pedazo página-resumen de Javascript, pero que ocupa varias páginas, con referencia sobre sintaxis, orientación a objetos, así como compatibilidad de distintos componentes de la tecnología con cada navegador.

Expresiones regulares en Javascript
Con explicaciones muy rápidas y esquemáticas sobre expresiones regulares.

Jquery Cheat Sheet
Una página con una referencia rápida para los que trabajan con Jquery 1.2.
Ver también esta otra chuleta a color de Jquery.

Prototype Javascript Library 1.5.0
Una superconcentrada hoja de clases y métodos de Prototype.

A field guide to scriptaculous combination effects
Referencia rápida a efectos con Scriptaculous.

Mootools 1,1 Cheat Sheet
Hoja de referencia a las clases de Mootools.

Chuleta DOM
Todos los métodos del DOM de Javascript con rápidas explicaciones.

Artículo por Miguel Angel Alvarez

7.3.- Por dónde continuar aprendiendo Javascript

Qué otras referencias y manuales existen para seguir con tu aprendizaje de Javascript.

En este artículo pretendemos ofrecer un epílogo al Manual de Javascript II y un repaso a otras referencias que existen en este momento dentro de DesarrolloWeb.com que nos pueden ayudar a seguir aprendiendo el lenguaje de programación Javascript y la manera de utilizarlo para realizar todo tipo de efectos e interactividades en la página.

Si has seguido hasta este punto los manuales de Javascript generales, es decir, el Manual de Programación en Javascript I y su segunda parte, el Manual de Programación en Javascript II, estamos seguros que tendrás ya un conocimiento sólido sobre este lenguaje y las posibilidades básicas de la programación en el cliente.

Sin embargo, tú mismo te puedes preguntar cómo hacer muchas de las cosas que ves en tantas y tantas páginas web, como interfaces de usuario avanzadas que responden a la interacción con el visitante, efectos especiales, Ajax, etc. Como podrás comprobar a continuación, todo esto se puede aprender sin salir de DesarrolloWeb.com y afortunadamente ahora está a tu alcance con poco esfuerzo adicional.

7.3.1.- Javascript a fondo

La primera referencia que te queremos comentar es la sección dedicada exclusivamente a Javascript dentro de DesarrolloWeb.com. Esto sería como una "portada" del sitio dedicada a Javascript, donde publicamos todos aquellos contenidos que tienen que ver con este lenguaje.

http://www.desarrolloweb.com/javascript/

7.3.2.- Taller de Javascript

En el taller de Javascript encontrarás diversos artículos prácticos sobre cómo hacer las cosas más variadas con Javascript. Puedes tratarlos como prácticas, para aprender a hacer una gama de utilidades que te darán una base adicional sobre el lenguaje, o puedes consultarlos cuando tengas que resolver un problema concreto con Javascript.

http://www.desarrolloweb.com/manuales/22/

7.3.3.- Otros manuales prácticos

Tenemos también diversos manuales eminentemente prácticos, sobre aspectos muy concretos que se utilizan habitualmente en Javascript. En estos manuales detallamos cosas como el trabajo con imágenes, formularios, frames, ventanas secundarias, etc. Muchas de estas cosas ya las empezamos a tratar en el presente manual, pero existe una información mucho más completa que quizás te interese leer llegado el momento.

7.3.4.- Frameworks Javascript

Los frameworks son como librerías de código para hacer tareas comunes en páginas web, creadas por otros programadores y que están a tu disposición para acelerar el proceso de creación de páginas realmente avanzadas. Digamos que cualquiera de nosotros podría programar a mano y desde cero cualquiera de las funcionalidades implementadas en los frameworks, pero ello le ocuparía mucho más tiempo y los resultados lo más probable es que fueran peores.

Los frameworks son sin duda el paso que diferencia el programador de Javascript básico y el programador profesional, sin límites más allá de los propios del navegador y su propia imaginación. Por ello, como te podrás imaginar,

recomendamos encarecidamente que aprendas uno de ellos para realizar aplicaciones web basadas en Javascript realmente profesionales.

El único problema que podrás encontrar es que utilizar un framework en muchas de las ocasiones no es una tarea trivial y requiere que el programador tenga bastante habilidad. Sin embargo, en DesarrolloWeb.com hemos tratado en profundidad algunos de los frameworks Javascript más populares y estamos seguros que con nuestros manuales podrás aprender todo lo que necesitas saber para cumplir tus objetivos... y más.

Además, los frameworks solucionan uno de los problemas más grandes que tiene Javascript (si no el más crucial) y es el hecho de que el lenguaje no es exactamente igual en los diversos navegadores del mercado y las versiones que han ido saliendo. Por ello, a medida que se complican las cosas, comprobarás que hay muchas tareas para las cuales tienes que detectar el navegador que está ejecutando la página y realizar acciones distintas dependiendo de ello. Con los frameworks este problema se resuelve y nunca más tendrás que preocuparte de que tu código se ejecute bien en todos los navegadores.

Hablando de librerías Javascript, debemos saber que existen muchas posibilidades que relatamos en el artículo Listado de los Distintos Frameworks Javascript. En DesarrolloWeb.com empezamos hace tiempo a explicar algunos de ellos y comenzamos por uno llamado Cross-browser, pero que realmente no recomendamos, porque se quedó poco actualizado. Podemos tenerlo en cuenta como un precursor de los frameworks Javascript, pero que se ha quedado en desuso. Actualmente existen opciones mucho mejores.

Sin lugar a dudas, jQuery es el framework Javascript más popular en estos momentos. A poco que se aprenda de jQuery se podrá comprobar que es una auténtica delicia para implementar tanto interactividad como efectos especiales sobre páginas web. Para aprender este framework tenemos un completo Manual de jQuery que no exageramos si decimos que es uno de los mejores manuales ya publicados en DesarrolloWeb.com. Además, hemos publicado también un Taller de jQuery para aprender por la práctica.

Otro de los frameworks populares es Mootools, casi tan bueno como jQuery, pero con sus particularidades. En mi opinión, resulta un poco más complicado de manejar para ciertos usos, pero es una gran librería. Para aprender puedes consultar el Manual de Mootools, tratado también con gran detalle y el Taller de Mootools.

También tenemos un tercer framework en discordia, que nos lo ofrece el equipo de Yahoo!. En esta ocasión el manual sólo tiene una ligera presentación y explicaciones para empezar a trabajar, en el Manual de Introducción a YUI.

7.3.5.- Videotutoriales de Javascript

Para quien lo prefiera, en el momento de escribir estas líneas estamos publicando varios materiales para aprender Javascript de una manera más visual. Para ello estamos produciendo unos videotutoriales que te explicarán cosas sobre el propio lenguaje y alguno de los frameworks más utilizados. Consulta el Videotutorial de Javascript y el Videotutorial de jQuery.

7.3.6.- Conclusión sobre los contenidos didácticos Javascript

En fin, que como se puede comprobar, en DesarrolloWeb.com tienes todo lo que necesitas para seguir aprendiendo mucho Javascript... Confiamos en que, con un poco de esfuerzo por tu parte, puedas aprovechar todos los materiales que venimos publicando, para llegar a dominar este lenguaje y sus diferentes aplicaciones.

En este artículo ni siquiera hemos nombrado todos los manuales existentes sobre Javascript en este sitio, porque son muchos y algunos de ellos son demasiado específicos. Pero en definitiva, para estar al tanto de todos los manuales que tenemos y podamos seguir publicando en el futuro sobre Javascript, te recomendamos echar un vistazo a la sección de Javascript a fondo.

Artículo por Miguel Angel Alvarez

7.4.- Un vistazo a los nuevos eventos Touch de JAVASCRIPT

Ese es el primero de una serie de artículos donde veremos y conoceremos una referencia sobre manejo de eventos Touch con JAVASCRIPT.

Con la llegada de HTML5 al mundo del desarrollo web, la forma de crear aplicaciones web ha cambiado de manera muy significativa, ahora bien, JAVASCRIPT no es HTML, pero sí es un elemento de gran importancia para todo lo que significa y se ha logrado con la quinta versión del lenguaje de etiquetas, donde los desarrolladores tienen a mano nuevos elementos. Estos son soportados por nuevas o mejoradas APIS JAVASCRIPT. Aunque no podemos olvidar que la versión tres de CSS ha contribuido en gran medida a una nueva forma versátil y adaptable de visualizar por parte de los usuarios y a crear aplicaciones web por parte de los desarrolladores y diseñadores, con una sutileza semántica.

Todo ha cambiado y JAVASCRIPT lo ha hecho, dando un amplio soporte a dispositivos móviles como smartphones y tablets, además de servir como pilar de nuevos y asombrosos elementos HTML como el CANVAS, donde se construye cualquier cantidad de gráficos a través de JAVASCRIPT, pero en estos momentos lo que nos interesa de las nuevas herramientas del popular lenguaje web del lado del cliente, es el soporte que da al manejo de eventos “Touch”, extendiendo las APIS que tienen como tarea manejar todo lo referente a este tema.

En el presente artículo estudiaremos los diferentes eventos Touch que podemos procesar o manejar con JAVASCRIPT, además de algunos otros aspectos que son de gran importancia para desarrollar aplicaciones que requieran manejos de eventos Touch. Es importante decir que èste es el primero de una serie de artículos, que tienen como fin u objetivo brindar un esquema de orientación donde veremos brevemente algunos usos que podemos dar a los eventos Touch.

7.4.1.- Tipos de eventos Touch

Los dispositivos táctiles trabajan sin ningún problema en cualquier aplicación web, sus navegadores móviles hacen que los eventos de ratón como el clic, terminen por convertirse en un evento Touch, por eso, al hacer una aplicación web, ésta funciona aunque sea procesando eventos del ratón. Ahora se han agregado a JAVASCRIPT una interesante gama de eventos Touch, que a pesar de ser solo de tres tipos, podemos hacer una combinación de los mismos, y así se obtienen mejores resultados. Estos tipos de eventos son:

- **touchstart**: Este se genera al hacer cualquier toque a la pantalla, sin importar su duración o movimientos realizados.
- **touchend**: Este se ejecuta una vez se deja de tocar la pantalla o el objeto que tiene asignado el manejador de eventos.
- **touchmove**: Este es ejecutado una vez se desliza o desplaza el dedo el usuario, por encima de la pantalla u objeto que está siendo controlado a través del manejador de eventos.

Son tres eventos, los cuales son muy simples de entender, con ellos obtenemos aplicaciones web móviles, que sean más dadas a estos entornos táctiles.

Nota: Es importante mencionar que estos tres eventos funcionan en dispositivos Touch, pero se pueden simular con algún que otro software y también son compatibles con el sistema Touch presente en el ratón de los MacBook Air de Apple.

7.4.2.- Herramientas adicionales de los eventos Touch

Hasta ahora ya sabemos que contamos con tres tipos de eventos Touch, éstos, a su vez, tienen una serie de elementos que complementan todo el manejo y procesamientos que se generan tras la interacción de un usuario a través de un dispositivo táctil.

Es importante mencionar que cada evento Touch posee una lista de propiedades en común que vendrían siendo el complemento del que se hablaba anteriormente. Hay tres propiedades que están ligadas de forma directa al Touch, que son:

- **touches**: Es una lista de todos toques que se han generado en la pantalla, tiene poca utilidad y suele ser poco usada.
- **targetTouches**: Éste guarda una lista de la cantidad del evento que se ha generado en un elemento del DOM.
- **changedTouches**: Éste guarda una lista de todos cambios que se producen hasta llegar al evento Touch, por ejemplo, en un touchsend puede haber un touchstart y un touchmove.

Hay otro grupo de propiedades encargado de guardar informacion sobre el evento, las cuales son:

- **identifier**: Un número único que identifica de forma única cada toque generado durante una sesión.
- **target**: El elemento del DOM en donde se generó el evento.
- **client/ page/ screen**: Información de la pantalla, relevante sobre acciones que genera el evento.
- **radius coordinates and rotationAngle**: Describe una aproximación de las elipses generadas.

7.4.3.- Procedimiento para asignar un evento Touch

Los eventos Touch de JAVASCRIPT no cambian para nada el esquema que normalmente se usa para crear manejadores en este lenguaje, así que para iniciar, no se está hablando de diferencias a la hora de asignar un manejador de eventos Touch, antes que cambiar los eventos se han extendido, así que ahora podemos asignar a cualquier elemento del DOM uno o varios de los tres eventos mencionados algunos párrafos anteriores de éste artículo.

Para facilitar un poco la explicación, veamos algo de código sencillo, iniciaremos por obtener la referencia a través de métodos del DOM.

```
//Obtenemos el elemento con el que vamos a trabajar
var elementoTouch= document.getElementById("areaTactil");
```

Esto no es nada desconocido para los entendidos de JAVASCRIPT, pero si no sabes a cerca del tema, puedes revisar los manuales de JAVASCRIPT que se encuentran a disposición en Desarrolloweb.com. Lo que viene a continuación es la asignación del manejador del evento, además de algunos datos sobre el elemento, como las coordenadas donde se ha llevado a cabo el toque de pantalla.

```
//posteriormente asignamos el manejador de eventos lo cual
// se hace de manera convencional.
elementoTouch.addEventListener('touchstart', function(event){
//Comprobamos si hay varios eventos del mismo tipo
if (event.targetTouches.length == 1) {
var touch = event.targetTouches[0];
// con esto solo se procesa UN evento touch
alert(" se ha producido un touchstart en las siguientes cordenas: X " + touch.pageX + " en Y " +
touch.pageY);
}

}, false);
```

En este caso se está manejando un solo evento. En próximos artículos veremos cómo manejar varios eventos, capacidad que solo se puede apreciar en dispositivos multi-Touch.

Artículo por Dairo Galeano

7.5.- Datos adicionales en el trabajo con Eventos Touch de JavaScript

Damos los últimos retoques en la triología de artículos dedicada a los eventos Touch de JavaScript. Multi Touch y mucho más.

En dos artículos ya nos hemos dado por enterado de cómo proceder ante el manejo de eventos Touch en aplicaciones Web móvil, un mercado que crece cada día. Por tal razón es que estos dispositivos avanzan de forma acelerada, con un crecimientos abrumador en variedad y avances de herramientas con las que cuentan teléfonos inteligentes y tabletas, que hacen uso de pantallas táctiles para la interacción directa con el usuario. La evolución ha llevado a la concepción de dispositivos multi-Touch, capaces de responder a varios eventos al tiempo, esto ha abierto la puerta a una nueva generación de aplicaciones que están dirigidas a estos sofisticados y avanzados dispositivos, y sólo es posible en tabletas y teléfonos inteligentes de última generación, con sistemas operativos que implementen esta tecnología, pues esta no solo depende del hardware.

Ante este escenario, JavaScript también se preparó para ser capaz de responder a eventos mult-Touch, haciéndolo de una forma muy sencilla, tal como se han venido manejando los eventos Touch en esta serie de artículos, siendo este el tercero, precedido de los artículos "Un vistazo a los nuevos eventos Touch de JavaScript" y "combinando Eventos Touch de JavaScript". En este donde se pretende rematar los apuntes finales dedicados de mi parte a los eventos Touch de JavaScript, veremos cómo hacer objetos arrastrables, haciendo uso de los eventos Touch, además de cómo procesar múltiples eventos Touch, dirigido a dispositivos avanzados, pero si tu terminal no tiene soporte multi-Touch, de igual forma el ejemplo funcionará, solo que con un solo eventos a la vez.

7.5.1.- Haciendo un objeto arrastrable gracias a los eventos Touch

Para engrosar un poco los ejemplos que hemos desarrollado en esta serie de artículos que hemos dedicado al estudio de los eventos Touch a través del lenguaje de la web JavaScript, vamos a crear un ejemplo que, aunque muy simple, nos dará a conocer una forma de hacer un objeto arrastrable. A continuación crearemos un elemento div el cual se podrá mover por la pantalla gracias al deslizamiento que haga el usuario. Para hacerlo posible vamos a usar el evento *touchmove*, haciendo muy fácil la implementación del ejemplo, pero bueno veamos el código que usamos para lograr eso:

```
var obj=document.gelElmentById("objArrastrable");
obj.addEventListener('touchmove', function(event){
    if (event.targetTouches.length == 1) {
       var touch = event.targetTouches[0];
       // con esto solo se procesa UN evento touch
       obj.style.left = touch.pageX + 'px';
       obj.style.top = touch.pageY + 'px';
    }
       }, false);
```

Es importante destacar que ese elemento debe tener algunas reglas de estilo CSS, para poder llevar a feliz término el desarrollo del ejemplo en mención. Inicialmente hay que poner la posición en absoluta del objeto, de lo contrario no se moverá por la pantalla, también debe tener en el estilo CSS los atributos que son modificados a través de los métodos del DOM, es decir, haciendo uso de JavaScript. Como se puede apreciar en el ejemplo, por mi parte hice un pequeño

estilo con el fin de poder llevar a cabo la ejecución del ejemplo, estilo que comparto con ustedes, aunque no está de más decir que no se hace realmente algo extraordinario.

```
#objArrastrable{
          border: #000 solid 2px;
          position: absolute;
          width:100px;
          height: 100px;
          top: 450px;
          left: 100px;
       }
```

Con este simple estilo, lo único que garantizamos es la ejecución del ejemplo y será decisión tuya si quieres agregar algo más para mejorar de forma visual tus ejemplos o proyectos que hagan uso de los eventos Touch de JavaScript.

7.5.2.- Haciendo un canvas dibujable por el usuario de forma multi Touch

Para quienes hayan aprendido programación Java, quizás hayan podio toparse con un ejemplo muy famoso: es un pequeño espacio donde, al hacer un arrastre con el ratón, se iba dibujando un punto. Este ejemplo aparece en un popular libro de programación, donde lo usaban para enseñar el procesamiento de eventos que se pueden generar con el ratón del ordenador. Traemos a colación ese viejo recuerdo, pues ahora haremos algo parecido, pero nosotros no estamos hablando de eventos del ratón de una computadora, sino de eventos Touch de un dispositivo móvil. Tampoco lo programaremos en Java, sino en JavaScript, usando el elemento canvas de HTML5.

> **Nota:** Es importante mencionar que, para entender el funcionamiento de este ejemplo, es vital saber manejar el elemento canvas de HTML5. Si no tienes idea acerca de esta nueva etiqueta, te invitamos a que indagues en el Manual de canvas, el cual está disponible en DesarrolloWeb.com.

El ejemplo realmente no hace nada diferente a lo explicado en esta serie de artículos, dedicados fielmente al manejo de eventos Touch. El único cambio drástico es que será capaz de responder a múltiples eventos a la vez, para lo cual creamos un código como el siguiente:

```
canvas.addEventListener('touchmove', function(event) {
    for (var i = 0; i < event.touches.length; i++) {
   var touch = event.touches[i];
    ctx.beginPath();
    ctx.arc(touch.pageX, touch.pageY, 20, 0, 2*Math.PI, true);
    ctx.fill();
    ctx.stroke();
    }
}, false);
```

Como se puede apreciar, procesar múltiples eventos es muy sencillo, sólo hay que hacer un ciclo que itere a través de la lista del objeto touches del evento; de esta forma respondemos a todos los eventos touchmove generados en el *canvas*. No está de más recordar que el funcionamiento de este ejemplo no está garantizado en todos los dispositivos. Su uso está garantizado en iPad dos, Motorola Xoom, donde se ha probado el funcionamiento de este ejemplo.

También compartimos el código completo del ejemplo, además de la opción de verlos en funcionamiento en una página aparte.

```
<!DOCTYPE html>
<html lang="es">
<head>
    <title>Estacando eventos Touch</title>
    <meta charset="utf-8" />
    <script>
       function inicia(){
          var obj=document.getElementById("objArrastrable");
```

```
           var canvas = document.getElementById('objeto');
           var ctx= canvas.getContext('2d');
obj.addEventListener('touchmove', function(event){
    if (event.targetTouches.length == 1) {
       var touch = event.targetTouches[0];
       // con esto solo se procesa UN evento touch
       obj.style.left = touch.pageX + 'px';
       obj.style.top = touch.pageY + 'px';
    }
       }, false);

   canvas.addEventListener('touchmove', function(event) {
    for (var i = 0; i < event.touches.length; i++) {
   var touch = event.touches[i];
    ctx.beginPath();
    ctx.arc(touch.pageX, touch.pageY, 20, 0, 2*Math.PI, true);
    ctx.fill();
    ctx.stroke();
    }
}, false);
   }
    </script>

    <style>
       #objArrastrable{
          border: #000 solid 2px;
          position: absolute;
          width:100px;
          height: 100px;
          top: 450px;
          left: 100px;
       }
    </style>
</head>
    <body onLoad="inicia();">
      <canvas width="450" height="350" style="border: #000 solid 3px; " id="objeto"></canvas>
      <div id="objArrastrable">
         Objeto arrastrable
      </div>
    </body>
</html>
```

Artículo por Dairo Galeano