0

JAVASCRIPT II

Realizado por Diana Alfaro

Contenido del Programa
Proyecto: Desarrollo avanzado de herramientas Web

- Compatibilidad del Navegador
- ¿Qué son las hojas de estilo?
- Importación
- Estilos
- Propiedades
- Jerarquías
- Manejo de Imágenes con Java Script
- Manejo del mouse
- Manejo de eventos
- Manejo del scroll
- Detección de botones del Mouse
- Manejo de captura de teclado
- Identificar las teclas presionadas
- Validación de formularios con eventos de teclado
- Control de propiedades de los elementos HTML
- Frames
- Tablas
- Fuentes
- Encabezados
- Botones
- El Objeto Eval
- El objeto Time
- Lectura de Cookies
- Cookies y JavaScript
- Obtención de información de una cookie
- Diseño de Objetos
- Manejo de errores

JAVA SCRIPT NIVEL 2

- ***COMPATIBILIDAD DEL NAVEGADOR***

JavaScript 1.6

Method/feature	Web browser support
	Array extras:
The indexOf method	2.0+ 3.0+ 1.0+ 9.5+
The lastIndexOf method	2.0+ 3.0+ 1.0+ 9.5+
The every method	2.0+ 3.0+ 1.0+ 9.5+
The filter method	2.0+ 3.0+ 1.0+ 9.5+
The forEach method	2.0+ 3.0+ 1.0+ 9.5+
The map method	2.0+ 3.0+ 1.0+ 9.5+
The some method	2.0+ 3.0+ 1.0+ 9.5+

- ***¿QUÉ SON LAS HOJAS DE ESTILO?***

Las Hojas de Estilo (o CSS, por Cascading StyleSheets) son un mecanismo que permiten aplicar formato a los documentos escritos en HTML (y en otros lenguajes estructurados, como XML) separando el contenido de las páginas de su apariencia. Para el diseñador, esto significa que la información estará contenida en la página HTML, pero este archivo no debe definir cómo será visualizada esa información. Las indicaciones acerca de la composición visual del documento estarán especificadas en el archivo de la CSS.

- ***CÓMO FUNCIONAN***

Veamos primero como trabaja el código HTML. En HTML, las etiquetas (tags) le indican al navegador los elementos que componen la página y éste aplica el formato a cada elemento en particular, por ejemplo:

<H1>Título</H1>

especifica que el texto "Título" es un Encabezado (Heading) de nivel 1 dentro de los 6 niveles definidos por HTML. El navegador aplicará a ese texto el formato predeterminado (que varía un poco si se trata de Internet Explorer, Netscape, o si usamos Windows, Mac o Linux).

Si quisiéramos componer los encabezados H1 con tipografía Arial, de 19 puntos, en color azul y alineación central, deberíamos especificarlo del siguiente modo:

```
<H1 ALIGN="center">
 <FONT FACE="Arial" COLOR="#0000FF" SIZE="5">
   Título</FONT>
</H1>
```

Por supuesto, esto debería repetirse en cada encabezado H1 de cada página de nuestro sitio.

- ***IMPORTACIÓN***

Las Hoja de Estilo no utilizan el archivo de la página Web para especificar el formato de la página (en realidad, a veces pueden hacerlo, como veremos más adelante). En su lugar, usan un archivo de texto puro con extensión .CSS que luego se vincula a la página.

Este archivo contiene reglas que constan de un selector (en este ejemplo, H1) y una o más declaraciones (en el ejemplo tenemos cuatro declaraciones). Cada declaración tiene dos partes: una propiedad (por ejemplo, FONT SIZE) y un valor (en este caso, 19pt). Estas reglas son las que determinan cómo deberá mostrarse la página.

Nuevamente, si quisiéramos componer los encabezados H1 con tipografía Arial, de 19 puntos, en color azul y alineación central, el archivo CSS debería contener el siguiente texto:

```
H1 {
  font-family: Arial, Sans-serif;
  font-size: 19pt;
  color: #0000FF;
  text-align: center;
}
```

Luego, en cada página de nuestro sitio agregamos un link a la Hoja de Estilo:

```
<LINK REL="stylesheet" HREF="hoja_de_estilo.css" TYPE="text/css">
```

El elemento LINK especifica:

1. el tipo de vínculo: a una hoja de estilo ("stylesheet")
2. la ubicación de la hoja de estilo a través del atributo "href"
3. el tipo de hoja de estilo que se vincula: "text/css"

Ahora, todos los encabezados H1 de las páginas que contienen la referencia al archivo de la CSS tendrán el aspecto que hemos definido.

HOJAS DE ESTILO INCRUSTADAS

Hemos visto cómo las reglas de estilo se especifican en un archivo externo. Este método es el más recomendable y el que permite mayor flexibilidad: los estilos pueden cambiarse sin tocar el código HTML y la CSS puede ser compartida por varias páginas.

También existe la posibilidad de poner la hoja de estilo dentro de una página HTML usando el elemento STYLE.

```
<HTML>
 <HEAD>
 <TITLE>CSS incrustada</TITLE>
 <STYLE type="text/css">
  H1 { color: blue }
 </STYLE>
 </HEAD>
 <BODY>
  <H1>Título color azul</H1>
  <P>Un párrafo cualquiera...
 </BODY>
</HTML>
```

Como se ve en el ejemplo, el elemento STYLE se usa dentro del encabezado (<HEAD></HEAD>) de la página, especificando el tipo de hoja de estilo:

```
<STYLE type="text/css">

  H1 { color: blue }

  ... (aquí se agregan todas las reglas de estilo) ...

</STYLE>
```

Este método permite aplicar la hoja de estilo solamente a la página que la contiene. Si bien no es tan práctico como usar una CSS externa, resulta útil cuando en nuestro sitio tenemos algunas pocas páginas que usan un formato distinto al resto.

En muchos casos convendrá usar ambos métodos simultáneamente: poner un link a una hoja externa para aplicar un estilo general y luego crear una hoja incrustada en la que solamente deberemos definir las reglas específicas para esa página. Por las leyes de cascada de las CSS, en caso de existir una misma regla (una en la hoja externa, otra en la hoja incrustada) con distintos valores, tiene preponderancia la definida en la hoja incrustada.

HOJAS DE ESTILO EN LÍNEA

Por último, también es posible aplicar el estilo directamente en la etiqueta HTML:

<P STYLE="text-align: left; text-indent: 1em">

Por supuesto, la definición del estilo dentro de la propia etiqueta (tag) no es la manera más eficaz de utilizar las hojas de estilo, pero pueden existir casos que lo justifiquen. La existencia de una regla como la del último ejemplo constituye por sí misma una hoja de estilo por lo que debemos declarar en el encabezado de la página el tipo de CSS que estamos usando.

<STYLE type="text/css"></STYLE>

Esta declaración no será necesaria si en la misma página estamos usando una hoja de estilo incrustada o si existe una referencia a una hoja externa.

Esta forma de aplicar la hoja de estilo directamente en la etiqueta es similar al modo usado en el código HTML, pero con dos importantes diferencias:

1. El conjunto de propiedades que se pueden aplicar es mucho mayor.
2. Los estilos especificados en una hoja de estilo (cualquiera sea su origen) tienen preferencia por sobre los formatos aplicados con HTML.

Esto significa que si en nuestra hoja de estilo hemos definido:

P {text-align: left}

Todos los párrafos quedarán alineados por la izquierda y, si en un párrafo determinado, usamos:

<P align="center">

El párrafo seguirá estando alineado por la izquierda porque las reglas de las CSS tienen más fuerza que el formato aplicado con HTML.

PROPIEDADES

Ahora que ya hemos visto cómo se definen estilos en un documento HTML, así como todas las posibilidades en cuanto a jerarquías, clases, nos vamos a centrar en qué es lo que podemos poner en cada una de esas parejas propiedad:valor que decíamos que definen un estilo.

Para facilitar su identificación, los vamos a dividir en las siguientes categorías:

Propiedes:

- Formato de bloque
- Las fuentes
- Texto
- Color y fondo
- Listas

Hay propiedades en las que necesitaremos especificar alguna longitud (por ejemplo, en los márgenes). Para ello, usaremos esta notación:

[-]NNtipo

- es el signo, y es opcional (es lo que quieren decir los corchetes, que se trata de algo opcional, los corchetes no hay que ponerlos).

NN es la cantidad, y tipo es la magnitud. Esta última puede ser relativa o absoluta.

Las magnitudes relativas son em (el alto de la M mayúscula), ex (la mitad de la altura de la M mayúscula, que viene a ser aproximadamente la altura de la x minúscula), px (píxel). Las magnitudes absolutas son pt (puntos), pc(picas), in (inches, es decir, pulgadas), mm(milímetros), cm (centímetros).

Hay otras propiedades en las que tendremos que especificar un color; para ello hay tres posibilidades: escribiéndolo de la misma forma que en HTML, con la notación #RRGGBB, siendo RR, GG, BB los valores en hexadecimal de las componentes roja, verde y azul del color, usando algún nombre predefinido, o usando la función rgb(R,G,B), donde R, G, B son los valores en decimal de las componentes roja, verde y azul del color.

Un último detalle a comentar antes de pasar al estudio de las propiedades y sus posibles valores, es el siguiente: desde el punto de vista de las hojas de estilo, existen tres tipos de elementos HTML: de **bloque** (que son los que hacen empezar línea nueva, como <P>, las cabeceras, ...), incrustados en **línea** (que no alteran la línea en la que se encuentran, como <B>, <I>, ...) y de **lista** (que son los elementos de una lista delimitados por <LI>). Todo elemento de bloque se considera rodeado por una caja, con propiedades de márgenes, borde, padding y fondo. Además, la caja que lo rodea tiene un cierto ancho, y una cierta alineación con respecto al documento.

VENTAJAS (Y DESVENTAJAS) DE LAS HOJAS DE ESTILO

Si consideramos los ejemplos anteriores, resultan evidentes algunas ventajas:

Con una Hoja de Estilo podemos alterar la presentación de cada elemento sin tocar el código HTML, ahorrando esfuerzo y tiempo de edición. Si quisiéramos alinear a la izquierda los encabezados H1 de nuestras páginas, bastaría con cambiar en la CSS la declaración "text-align: center" por "text-align: left" e inmediatamente cada H1 se alinearía a la izquierda en todas las páginas vinculadas a la Hoja de Estilo. De este modo no sólo simplificamos el mantenimiento del sitio sino que además reducimos las posibilidades de cometer errores.

El lenguaje de las CSS ofrece herramientas de composición más potentes que HTML. Hemos especificado en los ejemplos una fuente alternativa genérica (Sans-serif) para el caso de que la máquina del usuario no contenga la Arial (en HTML no existen estas fuentes genéricas). Con HTML, el tamaño de la fuente se especifica con un sistema de medidas predeterminadas por el browser (en el ejemplo, SIZE=5), con las CSS hemos especificado el tamaño en puntos tipográficos (y podemos hacerlo en cm, pixeles, cuadratines, altura de la x, etc.). Más aún, las CSS permiten aplicar

prácticamente todas las propiedades a cualquier elemento de la página, mientras que HTML sólo permite un número limitado de propiedades para cada elemento.

Se evita tener que recurrir a trucos para conseguir algunos efectos. Con CSS no es necesario usar imágenes invisibles para hacer una sangría (la propiedad text-indent se encarga de eso) o usar una tabla para ubicar un elemento en determinado lugar de la pantalla (las CSS permiten posicionar con precisión cualquier elemento).

Además:

El lenguaje de las Hojas de Estilo, aunque muy potente, es relativamente sencillo y fácil de aprender.

Los documentos que usan CSS generalmente resultan más compactos.

Las Hojas de Estilo pueden aplicarse de varias maneras y combinarse formando una cascada de estilos con la información de cada una.

Pueden usarse con otros lenguajes de programación (como JavaScript) para conseguir efectos dinámicos en las páginas.

Se pueden especificar Hojas de Estilo para diferentes navegadores y tipos de medios (impresos, braille, auditivos, etc.).

El usuario con alguna discapacidad (o simplemente por preferencias) puede definir su propia Hoja de Estilo y la regla !important obliga a su navegador a suplantar la Hoja de Estilo del autor.

Por cierto, existen otras ventajas muy importantes pero, como esto ya se parece a un folleto de ventas, terminaré aquí la lista de elogios. Ver

En cuanto a las desventajas en el uso de las Hojas de Estilo, la única de importancia es el soporte irregular que tienen las CSS por parte de los navegadores. Ciertas propiedades que funcionan en un browser no funcionan en otros, o existen diferencias en un mismo navegador según sea para Windows o Mac. También existen diferencias entre distintas versiones de un mismo browser.

Esto puede provocar que:

Nuestra página sea visualizada por el lector con un formato no deseado por nosotros. En todo caso, el navegador aplicará el formato predeterminado y nuestro trabajo de composición habrá sido inútil.

Algunas propiedades de las CSS (como las que afectan la posición o visibilidad de los elementos) pueden provocar que una parte del contenido de nuestra página resulte inaccesible desde ciertos navegadores si no son utilizadas correctamente.

Debe entenderse que las Hojas de Estilo fueron diseñadas para permitir que los autores influyan en la composición de la página, pero no para que la controlen. Una CSS sugiere al browser un estilo de composición para el documento pero no puede forzarlo a aplicar un formato determinado.

Las Hojas de Estilo son una herramienta que puede resultar muy efectiva para lograr una presentación atractiva de la página siempre que la página no sea dependiente

de la Hoja de Estilo. Se debe considerar en todo momento aquellos navegadores que no soportan CSS, cuidando que los mismos puedan mostrar la página correctamente y en su totalidad aún cuando nuestras reglas de estilo no sean aplicadas.

En este sentido, espero que esta guía de uso de las CSS ayude no sólo a conocer el lenguaje de las Hojas de Estilo sino también a aplicarlas con el mayor beneficio.

- ***JERARQUÍAS***

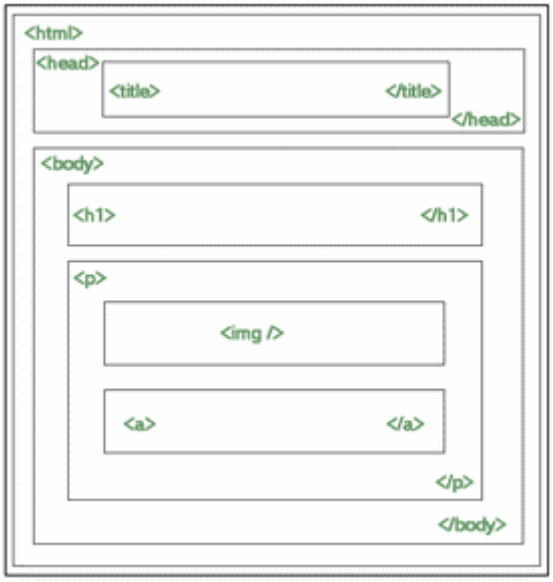
Cualquier documento HTML contiene un buen número de elementos: encabezados, párrafos, listas y demás. Muchas veces, los desarrolladores utilizamos el término "etiqueta" (tag) para referirnos a un elemento, hablando por ejemplo de la etiqueta . Pero la etiqueta es sólo la parte de un elemento. En realidad el elemento completo es Contenido del párrafo . Es decir al hablar de un elemento nos referimos a la etiqueta y a su contenido, que debemos tener en cuenta, puede contener a su vez otros elementos.

Lo que hay que tener claro es que cada elemento de un documento HTML está contenido en otro elemento y puede contener elementos dentro de él. Esto es lo que se llama en inglés "containment hierarchy" (jerarquía de elementos) de una página.

En el punto más alto de la jerarquía está el elemento <HTML> de la página. Todos los demás elementos de un documento html están contenidos dentro de este elemento. Así unos elementos están contenidos dentro de otros. Los elementos de párrafo, pueden contener imágenes o enlaces y a su vez están contenidos en el elemento body.

A la hora de entender bien el funcionamiento de las hojas de estilo y el concepto de herencia y cascada, debemos tener muy claro este tema de que los elementos se incluyen unos dentro de otros.

Gráficamente podríamos representar esto así:



Hemos dicho que entender esto es muy importante y la razón de ello, es que en muchos casos (y con css2 siempre que así lo queramos), los elementos pueden heredar propiedades del elemento que los contiene.

En esta jerarquía hablamos de padres, hijos y hermanos:

Un elemento A dentro del cual está un elemento B, es el padre del elemento B, y el elemento B es el hijo del elemento A. Por ejemplo.

El elemento <p> está contenido dentro del elemento <body>. Así que el elemento p es hijo del elemento body y puede heredar sus propiedades. El elemento body es el padre del elemento p.

Dos elementos con un padre común son hermanos. Por ejemplo, el elemento p contiene los elementos img, y a, que son hermanos entre sí e hijos ambos de p.

Es decir, supongamos que hemos establecido un tipo de fuente para el elemento body (arial por ejemplo), si al elemento p no le establecemos una fuente distinta, heredará la fuente de su padre y se mostrará también con el tipo de fuente Arial.

COMO FUNCIONA LA “CASCADA”

Una hoja de estilo asociada con uno o más documentos html nos puede resultar muy útil. Pero la verdadera potencia de de las hojas de estilo es la posibilidad de enlazar unas hojas de estilo con otras, creando un grupo de hojas de estilo para manejar sitios grandes. O simplemente sitios con secciones distintas que queremos diferenciar visualmente.

De esta forma podemos por ejemplo, establecer una hoja de estilo base, para todo un web site, que contenga por ejemplo el tipo de fuente, el tamaño, cómo se muestran las listas etc, y luego nos podríamos hacer una serie de hojas estilo más específicas para cada una de nuestras secciones.

Así podemos enlazar diversas hojas de estilo relacionadas, creando una jerarquía de hojas de estilo. Y esto es lo que se llama cascada, en seguida veremos porqué.

Este mecanismo de la cascada funciona mediante la posibilidad que tiene una hoja de estilo de "importar" otra hoja de estilo, formando así una "cascada" de hojas de estilo.

Así como hemos visto en el gráfico, podemos tener una hoja de estilo central y hojas de estilos por secciones que importan la información de la hoja de estilo central y añaden las propiedades que ellos necesitan. De esta forma, la actualización resulta fácil, puesto que si cambiamos una hoja de estilo "padre", sus hijas lo heredarán inmediatamente.

LA ESPECIFICIDAD

Tanto dentro de una hoja de estilos, como en un conjunto de hojas de estilo funcionando en cascada, puede ocurrir que más de una regla sea aplicable al mismo elemento. ¿Qué ocurre cuando dos propiedades, especificadas en reglas distintas y contradictorias entre sí son aplicables al mismo elemento? Obviamente no pueden aplicarse las dos (un texto no puede ser a la vez rojo y verde, por poner un ejemplo). Las reglas de especificidad son el mecanismo mediante el cual se resuelven estos conflictos. La regla general es que en caso de conflicto se aplica la regla más específica.

Algunos selectores, son más específicos que otros. Por ejemplo, los selectores de ID y de clase son más específicos que los selectores de etiqueta.

A la hora de tener en cuenta la especificidad, nos vendrá bien recordar lo siguiente:

1. Los selectores de ID son más específicos que los otros selectores.
2. Los selectores de clase son más específicos que los selectores de etiqueta
3. Los selectores contextuales y otros selectores que implican a más de un elemento, son más específicos que los selectores simples. (cuantos más elementos haya implicados, más específico será el selector y por tanto tendrá preferencia).

En los casos en que dos reglas tengan exactamente el mismo nivel de especificidad, tendrá preferencia el que esté después en la cascada.

Por ejemplo si tenemos una hoja de estilos A y una hoja de estilos B desde la que importamos las reglas de la hoja de estilos A. Si hay dos reglas contradictorias, prevalecerá la establecida en la hoja de estilos B por estar después en la cascada.

Si en una misma hoja de estilos hay dos reglas contradictorias, se aplicará la que esté en último lugar, la más alejada del comienzo del documento.

Aunque en principio todas estas reglas nos puedan parecer complicadas, la verdad es que aplicando el sentido común, no nos encontraremos con problemas de herencias y especificidad.

- ***MANEJO DE IMÁGENES CON JAVASCRIPT***

El lenguaje JavaScript posee un objeto propio asociado a cada una de las imágenes de un documento HTML, el objeto Image. Además, también posee un array particular, el array images, que contiene todas las imágenes presentes en la página.

Dentro de la jerarquía propia de este lenguaje, tanto el objeto Image como la matriz images se encuentran incluidos dentro del objeto document, que a su vez se encuentra incluido en al objeto principal window.

Por lo tanto, al ser estos objetos elementos propios del lenguaje, podemos referirnos a ellos y acceder a sus propiedades de forma directa, sin tener que recurrir a capas ni otros elementos externos. Así, podemos acceder directamente a una propiedad cualquiera de una imagen del documento de dos formas diferentes:

1) Mediante el objeto Image, siendo la sintaxis general en este caso la siguiente:

document.nombre_imagen.nombre_propiedad

donde nombre_imagen es el nombre asignado a la imagen mediante su atributo NAME (es condición indispensable para usar este método el haber asignado a la imagen un nombre identificador único mediante el atributo NAME). nombre_propiedad define la propiedad a la que queremos acceder.

2) Mediante la matriz images[]. Esta matriz contiene todas las imágenes del documento, empezando su índice interno por 0, como en todos los arrays de JavaScript. La sintaxis general es del tipo:

document.images[indice].nombre_propiedad

La equivalencia entre indice y la imagen que le corresponde se establece de forma secuencial, de tal forma que images[0] representará a la primera imagen insertada en el BODY de la página, images[1] a la segunda, y así sucesívamente, salvo que se haga una declaración explícita al respecto.

El array images posee la propiedad length, que almacena el número de imágenes presentes en el documento. Para obtener éste, basta con escribir:

document.images.length.

Ejemplos:

Acceso a la propiedad WIDTH de la imagen bandera:

document.images.bandera.width

o bien,

document.bandera.width

Acceso a la propiedad SRC de la imagen bandera:

document.images[1].src

Nota: el atributo width es de sólo lectura para Nestcape 4x, por lo que podemos leer su valor, pero no modificarlo. En cambio, sí podemos hacer esto en Internet Explorer y en Nestcape 6x.

El objeto Image posee una serie de propiedades, a las que podemos acceder mediante JavaScript, permitiendo este lenguaje leer estas propiedades e incluso cambiar muchas de ellas. La forma de acceder a todas ellas es:

document.nombre_imagen.propiedad

Las principales propiedades del objeto Image son:

name, que referencia el nombre identificador único de la imagen asociada al objeto. La forma de acceder a esta propiedad es:

document.images[indice].name

Podemos por ejemplo aprovechar el array images para obtener el name de todas las imágenes de nuestra página:

```
<script language="JavaScript" type="text/javascript">
        var nombres = "";
        for (i=0;i<document.images.length;i++)
        {
          nombres= + document.images[i].name + "/n";
        }
</script>
<input type="button" value="dame names" onClick="alert(nombres);">
```

El código JavaScript anterior debe situarse al final de la página, antes de la etiqueta </BODY>, para que las imágenes estén situadas antes que él, con objeto de que sepa las imágenes que hay en la página. Si lo establecemos tal como está dentro de la cabecera, al no haberse cargado todavía las imágenes, obtendremos la variable nombres como una cadena vacía. Para solucionar esto, podemos declarar antes de la variable las imágenes de la página, mediante el constructor de objetos Image, que veremos en el capítulo siguiente.

Como resultado de esta acción, sólo aparecerán aquellas imágenes a las que hemos nombrado por medio del atributo name.

También observarás, si pulsas el botón anterior, que en la ventana que aparece quedan muchos huecos vacíos; éstos son debidos a todas las imágenes de la página por encima del código a las que no hemos asignado la propiedad name.

src, que almacena la ruta de la imagen asociada al objeto. Así, si queremos asignar la ruta de una imagen a una variable, para poder por ejemplo presentar en pantalla esta ruta, deberemos escribir:

var ruta = document.nombre_imagen.src;

y luego podemos usar esta variable desde un botón de formulario, por ejemplo:

<img name="dinosaurio" src="images/avatar.gif" width="60" height="60" border="0">

```
<script language= "JavaScript" type="text/javascript">
        var ruta = document.dinosaurio.src;
</script>
<input type="button" value="dame ruta" onClick="alert(ruta);">
```

Esta propiedad es fundamentalmente usada para construir rollovers, en los que cambiamos dinámicamente su valor. Para ello, hay que establecerlo siempre entre comillas, dobles o simples, ya que src trata la ruta como una cadena de texto (variable de tipo string).

width / height, que representan la anchura y la altura de la imagen asociada al objeto, y que podemos leer (en los 3 navegadores comunes) y cambiar (sólo en Explorer y Nestcape 6x) de forma dinámica. Ejemplo:

<input type="button" value= "dame altura" onClick="alert(document.dinosaurio.height)">

border, que se refiere al atributo border de la imagen asociada al objeto. Para acceder a esta propiedad deberemos escribir:

document.images[indice].border o document.nombre_imagen.border

Como ejemplo, vamos a cambiar dinámicamente el borde de la imagen inferior:

<input type="button" value="cambia borde" onClick="document.icono.border=10;">

hspace / vspace, que definen el espacio horizontal o vertical de una imagen flotante respecto al texto que la rodea. El acceso a esta propiedad se logra mediante:

document.nombre_imagen.hspace

lowsrc, que fija la ruta de una imagen accesoria, generalmente una versión de menos peso en Ks de la imagen principal, que se debe cargar en pantalla mientras se recibe ésta. Su sintaxis es:

document.nombre_imagen.lowsrc

prototype: propiedad muy importante desde el punto de vista del programador web, ya que permite añadir propiedades y métodos adicionales a un objeto Image, es decir, permite ampliar lo que es el objeto en sí, aumentando sus propiedades por defecto. No vamos a entrar en detalles sobre esta propiedad, común a muchos de los objetos propios de JavaScript, ya que para su uso hace falta un conocimiento profundo de este lenguaje.

Pero si decir que sirve, por ejemplo, para asignar una nueva propiedad, que vamos a llamar orden, de tipo numérico, en la que podemos guardar un orden de presentación en una ventana flotante, y mediante código permitir al usuario que seleccione una serie de imágenes de las contenidas en nuestra página. Luego, mediante esta nueva propiedad orden podemos presentarlas ordenadas en la pantalla flotante (es sólo un ejemplo). Para más información al respecto, consultar un buen manual de JavaScript avanzado.

complete: propiedad del objeto Image de tipo booleano, cierta cuando la imagen se ha acabado de cargar en memoria, y falsa en caso contrario. Sólo es soportada por Nestcape Navigator, por lo que no se debe usar, en vistas a la compatibilidad.

- ***MANEJO DEL MOUSE***

En este grupo se pueden incluir los siguientes eventos:

onMouseOver: activado cuando el ratón se mueve por encima de un objeto;

onMouseOut: activado cuando el ratón sale de un objeto;

onMouseMove: se mueve el cursor del ratón, pero dado que esto sucede a menudo (el uso del ratón es frecuente), no está disponible como opción por defecto, sino tan sólo unida a la captura de eventos, que se explicará a continuación.

Los eventos onMouseOver y onMouseOut son complementarios ya que el primero se activa en el momento en que el cursor se coloca en el área del objeto cuya marca contiene el evento, mientras que el segundo lo hace cuando sale.

Los eventos onMouseOver y onMouseOut adquieren de la versión 1.0 a la 1.1 de Javascript la capacidad de estar asociados a la marca AREA, por lo que puede operar también con los mapas clicables; para Netscape, sin embargo, debe estar asociado también a la marca HREF, es decir, a un enlace, aunque ficticio.

MARCAS SENSIBILES

Otro límite es el impuesto a Netscape y las viejas versiones de Explorer por las marcas a las que el evento puede ser asociado:

Evento	Marcas asociadas en Netscape e JScript
onMouseOver	Este gestor se usa con los **botones de envío (submit)**, **pulsanti de reset (reset)**, **casillas de control (checkbox e radeo)**, **botones**, marcas <*INPUT*> de tipo OPTION y marca <*A*>.
onMouseOut	Usado con las marcas <**BODY**> y <**A**>
onMouseMove	Usado con los **botones** y las marcas <**BODY**> y <**A**>

ROLLOVER

De enorme importancia es el evento onMouseOver combinado con onMouseOut para crear el efecto RollOver.

La sintaxis es muy sencilla:

<A HREF="#" onmouseover="document.images[num].src='imagen.gif'" onmouseout=document.images[num].src='imagen1.gif'">

donde la almohadilla sustituye a cualquier otro enlace, mientras que num es el número de índice de la imagen en la página HTML.

Hace algunos años, cuando no existían programas como Flash, el rollover era el efecto gráfico de mayor difusión y seguramente el más apropiado para convertir en dinámico un sitio y animar elementos estáticos como los menús y las barras de navegación.

Como ejemplo, inserto un rollover algo complejo, el que puede activarse con un mapa clicable. Hay que destacar que para funcionar en Netscape hace falta añadir en cualquier caso un enlace al área sensible, resaltada en rojo (incluso si ha sido sustituido por el signo almohadilla), mientras que en Explorer se puede omitir.

<area shape="rect" coords="2,2,59,26" onmouseover="document.images[num].src='images/tre.gif'" href="#">

- **EVENTOS DEL TECLADO**

En este grupo se pueden incluir los siguientes eventos:

1. **onKeyPress**: evento que se activa cuando un usuario pulsa y deja de pulsar una tecla o también cuando la mantiene pulsada;
2. **onKeyDown**: activado cuando se pulsa la tecla;
3. **onKeyUp**: activado cuando una tecla, que se había pulsado, deja de pulsarse;
4. **onHelp**: activado cuando se pulsa la tecla F1;

El último evento se ha insertado en este sector aun cuando se activa sólo mediante la tecla F1.

Cada tecla tiene asignado un número en javascript, algunos de estos son:

Tecla	**Código**	**Tecla**	**Código**	**Tecla**	**Código**
backspace	8	tab	9	enter	13
shift	16	ctrl	17	alt	18
pause/break	19	caps lock	20	escape	27
page up	33	page down	34	end	35
home	36	left arrow	37	up arrow	38
right arrow	39	down arrow	40	insert	45
delete	46	0	48	1	49
2	50	3	51	4	52
5	53	6	54	7	55
8	56	9	57	a	65
b	66	c	67	d	68
e	69	f	70	g	71
h	72	i	73	j	74

k	75	l	76	m	77
n	78	o	79	p	80
q	81	r	82	s	83
t	84	u	85	v	86
w	87	x	88	y	89
z	90	left window key	91	right window key	92
select key	93	numpad 0	96	numpad 1	97
numpad 2	98	numpad 3	99	numpad 4	100
numpad 5	101	numpad 6	102	numpad 7	103
numpad 8	104	numpad 9	105	multiply	106
add	107	subtract	109	decimal point	110
divide	111	f1	112	f2	113
f3	114	f4	115	f5	116
f6	117	f7	118	f8	119
f9	120	f10	121	f11	122
f12	123	num lock	144	scroll lock	145
semi-colon	186	equal sign	187	comma	188
dash	189	period	190	forward slash	191
grave accent	192	open bracket	219	back slash	220
close braket	221	single quote	222		

Ya conociendo los números correspondientes a cada tecla solo nos basta con trabajar con Eventos en JavaScript, por ejemplo si queremos ver que tecla presiono en un input el código seria el siguiente:

<input type="text" onkeydown="tecla(event);" />

y el Javascript

```
<script type="text/javascript">
function tecla (e)
{
   var evt = e ? e : event;
   var key = window.Event ? evt.which : evt.keyCode;
   alert (key);
}
</script>
```

Podemos hacer un if y dependiendo de la tecla (según su key) realizar alguna acción, como llamar una función por ejemplo o no permitir el ingreso en el input de

caracteres no permitidos, por ejemplo si queremos que en un input no ingresen letras sino solo números, el código y su función seria algo así:

El input:

<input type="text" onkeypress="return num(event);" />

El Javascript:

```
<script type="text/javascript">
function num(e) {
   evt = e ? e : event;
   tcl = (window.Event) ? evt.which : evt.keyCode;
   if ((tcl < 48 || tcl > 57) && (tcl != 8 && tcl != 0 && tcl != 46))
   {
      return false;
   }
   return true;
}
</script>
```

LIBRERIA DE FUNCIONES 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.

EJEMPLOS DE FUNCIONES DE LA LIBRERÍA JAVASCRIPT

- **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.

FUNCIÓN PARSEINT

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.

- **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.

- **CLASE DATE EN JAVASCRIPT**

Se utiliza en Javascript para el manejo de fechas y horas, comentando sus métodos y propiedades.

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.

- **JERARQUÍA EN JAVASCRIPT**

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.

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.

NAVEGACIÓN A TRAVÉS DE LA JERARQUÍA

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.

OBJETO WINDOW DE JAVASCRIPT

Nos sirve para controlar la ventana del navegador.

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.

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.

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.

MÉTODOS DE WINDOW EN JAVASCRIPT

El objeto window de Javascript tiene a disposición de los programadores una larga lista de métodos.

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.

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)

EJEMPLOS DE MÉTODOS DE WINDOW

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

- **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")
```

- **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.

- **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.

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 objet frame en caso de que la página se esté mostrando en un frame.

PROPIEDADES DEL OBJETO DOCUMENT

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.

MÉTODOS DE DOCUMENT

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.

- **MANEJO DE EVENTOS**

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.

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>
```

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.

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>
```

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.

LOS MANEJADORES DE EVENTOS EN JAVASCRIPT

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 nose produce hasta que el usuario retira el foco de la aplicación del elemento. 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 elemento 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

MANEJADORES DE EVENTOS EN 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".

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 (onsub)mit: 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.

- **CLÁUSULAS TRY … CATCH: DETECTAR Y CAZAR ERRORES EN JAVASCRIPT**

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)
}
}
```

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
}c
atch (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.

- **VALIDACIÓN DE FORMULARIOS CON EVENTOS DE TECLADO**

La principal utilidad de JavaScript en el manejo de los formularios es la validación de los datos introducidos por los usuarios. Antes de enviar un formulario al servidor, se recomienda validar mediante JavaScript los datos insertados por el usuario. De esta

forma, si el usuario ha cometido algún error al rellenar el formulario, se le puede notificar de forma instantánea, sin necesidad de esperar la respuesta del servidor.

Notificar los errores de forma inmediata mediante JavaScript mejora la satisfacción del usuario con la aplicación (lo que técnicamente se conoce como "mejorar la experiencia de usuario") y ayuda a reducir la carga de procesamiento en el servidor.

Normalmente, la validación de un formulario consiste en llamar a una función de validación cuando el usuario pulsa sobre el botón de envío del formulario. En esta función, se comprueban si los valores que ha introducido el usuario cumplen las restricciones impuestas por la aplicación.

Aunque existen tantas posibles comprobaciones como elementos de formulario diferentes, algunas comprobaciones son muy habituales: que se rellene un campo obligatorio, que se seleccione el valor de una lista desplegable, que la dirección de email indicada sea correcta, que la fecha introducida sea lógica, que se haya introducido un número donde así se requiere, etc.

A continuación se muestra el código JavaScript básico necesario para incorporar la validación a un formulario:

```
<form action="" method="" id="" name="" onsubmit="return validacion()"> ...
</form>
```

Y el esquema de la función validacion() es el siguiente:

```
function validacion() {
  if (condicion que debe cumplir el primer campo del formulario) {
    // Si no se cumple la condicion...
    alert('[ERROR] El campo debe tener un valor de...');
    return false;
  }
  else if (condicion que debe cumplir el segundo campo del formulario) {
    // Si no se cumple la condicion...
    alert('[ERROR] El campo debe tener un valor de...');
    return false;
  }
  ...
  else if (condicion que debe cumplir el último campo del formulario) {
    // Si no se cumple la condicion...
    alert('[ERROR] El campo debe tener un valor de...');
    return false;
  }

  // Si el script ha llegado a este punto, todas las condiciones
  // se han cumplido, por lo que se devuelve el valor true
  return true;
}
```

El funcionamiento de esta técnica de validación se basa en el comportamiento del evento onsubmit de JavaScript. Al igual que otros eventos como

onclick y onkeypress, el evento onsubmit varía su comportamiento en función del valor que se devuelve.

Así, si el evento onsubmit devuelve el valor true, el formulario se envía como lo haría normalmente. Sin embargo, si el evento onsubmit devuelve el valor false, el formulario no se envía. La clave de esta técnica consiste en comprobar todos y cada uno de los elementos del formulario. En cuando se encuentra un elemento incorrecto, se devuelve el valor false. Si no se encuentra ningún error, se devuelve el valor true.

Por lo tanto, en primer lugar se define el evento onsubmit del formulario como:

onsubmit="return validacion()"

Como el código JavaScript devuelve el valor resultante de la función validacion(), el formulario solamente se enviará al servidor si esa función devuelve true. En el caso de que la función validacion() devuelva false, el formulario permanecerá sin enviarse.

Dentro de la función validacion() se comprueban todas las condiciones impuestas por la aplicación. Cuando no se cumple una condición, se devuelve false y por tanto el formulario no se envía. Si se llega al final de la función, todas las condiciones se han cumplido correctamente, por lo que se devuelve true y el formulario se envía.

La notificación de los errores cometidos depende del diseño de cada aplicación. En el código del ejemplo anterior simplemente se muestran mensajes mediante la función alert() indicando el error producido. Las aplicaciones web mejor diseñadas muestran cada mensaje de error al lado del elemento de formulario correspondiente y también suelen mostrar un mensaje principal indicando que el formulario contiene errores.

Una vez definido el esquema de la función validacion(), se debe añadir a esta función el código correspondiente a todas las comprobaciones que se realizan sobre los elementos del formulario. A continuación, se muestran algunas de las validaciones más habituales de los campos de formulario.

VALIDAR UN CAMPO DE TEXTO OBLIGATORIO

Se trata de forzar al usuario a introducir un valor en un cuadro de texto o textarea en los que sea obligatorio. La condición en JavaScript se puede indicar como:

```
valor = document.getElementById("campo").value;
if( valor == null || valor.length == 0 || /^\s+$/.test(valor) ) {
  return false;
}
```

Para que se dé por completado un campo de texto obligatorio, se comprueba que el valor introducido sea válido, que el número de caracteres introducido sea mayor que cero y que no se hayan introducido sólo espacios en blanco.

La palabra reservada null es un valor especial que se utiliza para indicar "ningún valor". Si el valor de una variable es null, la variable no contiene ningún valor de tipo objeto, array, numérico, cadena de texto o booleano.

La segunda parte de la condición obliga a que el texto introducido tenga una longitud superior a cero caracteres, esto es, que no sea un texto vacío.

Por último, la tercera parte de la condición (/^\s+$/.test(valor)) obliga a que el valor introducido por el usuario no sólo esté formado por espacios en blanco. Esta comprobación se basa en el uso de "expresiones regulares", un recurso habitual en cualquier lenguaje de programación pero que por su gran complejidad no se van a estudiar. Por lo tanto, sólo es necesario copiar literalmente esta condición, poniendo especial cuidado en no modificar ningún carácter de la expresión.

VALIDAR UN CAMPO DE TEXTO CON VALORES NUMÉRICOS

Se trata de obligar al usuario a introducir un valor numérico en un cuadro de texto. La condición JavaScript consiste en:

```
valor = document.getElementById("campo").value;
if( isNaN(valor) ) {
  return false;
}
```

Si el contenido de la variable valor no es un número válido, no se cumple la condición. La ventaja de utilizar la función interna isNaN() es que simplifica las comprobaciones, ya que JavaScript se encarga de tener en cuenta los decimales, signos, etc.

A continuación se muestran algunos resultados de la función isNaN():

```
isNaN(3);          // false
isNaN("3");        // false
isNaN(3.3545);     // false
isNaN(32323.345);  // false
isNaN(+23.2);      // false
isNaN("-23.2");    // false
isNaN("23a");      // true
isNaN("23.43.54"); // true
```

VALIDAR UNA DIRECCIÓN DE EMAIL

Se trata de obligar al usuario a introducir una dirección de email con un formato válido. Por tanto, lo que se comprueba es que la dirección parezca válida, ya que no se comprueba si se trata de una cuenta de correo electrónico real y operativa. La condición JavaScript consiste en:

valor = document.getElementById("campo").value;

```
if( !(/\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)/.test(valor)) ) {
  return false;
}
```

La comprobación se realiza nuevamente mediante las expresiones regulares, ya que las direcciones de correo electrónico válidas pueden ser muy diferentes. Por otra parte, como el estándar que define el formato de las direcciones de correo electrónico es muy complejo, la expresión regular anterior es una simplificación. Aunque esta regla valida la mayoría de direcciones de correo electrónico utilizadas por los usuarios, no soporta todos los diferentes formatos válidos de email.

VALIDAR UNA FECHA

Las fechas suelen ser los campos de formulario más complicados de validar por la multitud de formas diferentes en las que se pueden introducir. El siguiente código asume que de alguna forma se ha obtenido el año, el mes y el día introducidos por el usuario:

La función Date(ano, mes, dia) es una función interna de JavaScript que permite construir fechas a partir del año, el mes y el día de la fecha. Es muy importante tener en cuenta que el número de mes se indica de 0 a 11, siendo 0 el mes de Enero y 11 el mes de Diciembre. Los días del mes siguen una numeración diferente, ya que el mínimo permitido es 1 y el máximo 31.

La validación consiste en intentar construir una fecha con los datos proporcionados por el usuario. Si los datos del usuario no son correctos, la fecha no se puede construir correctamente y por tanto la validación del formulario no será correcta.

VALIDAR UN NÚMERO DE TELÉFONO

Los números de teléfono pueden ser indicados de formas muy diferentes: con prefijo nacional, con prefijo internacional, agrupado por pares, separando los números con guiones, etc.

El siguiente script considera que un número de teléfono está formado por nueve dígitos consecutivos y sin espacios ni guiones entre las cifras:

```
valor = document.getElementById("campo").value;
if( !(/^\d{9}$/.test(valor)) ) {
  return false;
}
```

Una vez más, la condición de JavaScript se basa en el uso de expresiones regulares, que comprueban si el valor indicado es una sucesión de nueve números consecutivos.

VALIDAR QUE UN CHECKBOX HA SIDO SELECCIONADO

Si un elemento de tipo checkbox se debe seleccionar de forma obligatoria, JavaScript permite comprobarlo de forma muy sencilla:

```
elemento = document.getElementById("campo");
if( !elemento.checked ) {
  return false;
}
```

- **FRAMES**

Cuando definimos una página que usará frames, dividimos la ventana actual en un conjunto de subpantallas que contienen cada una diversas páginas de HTML (inclusive provenientes de distintos sitios de Internet), cuyo tamaño está dado como un porcentaje de la ventana total o como una distancia en pixeles (algo muy semejante a la definición de tablas). Al realizar esto, el desarrollador puede buscar dos cosas: la primera es presentar de forma concisa y compacta una gran cantidad de información; la segunda, es la posibilidad de interactuar y obtener resultados (en una sola ventana) utilizando diferentes páginas HTML al mismo tiempo. Esto significa que los frames pueden intercambiar información el uno con el otro. Seguramente usted ya habrá visto páginas que usan frames debido a que una de sus aplicaciones más comunes son las relacionadas al establecimiento de una página de "resultados" y junto a ella (en alguno de los ejemplos por lo regular) existe otra página con botones o ligas que actúan como menú principal. De esta forma, usted puede seguir "navegando" mientras que conserva una serie de recursos disponibles todo el tiempo.

RECORDANDO LA CREACIÓN DE FRAMES CON HTML

Como usted debe comprender, no es el propósito de este manual el describir detalladamente la estructura y funcionamiento de la etiqueta FRAMES; sin embargo, es necesario repasar sus características para comprender mejor la forma en que JavaScript puede interactuar con él.

Para utilizar los frames de HTML, necesitamos de dos etiquetas importantes:

a) <FRAMESET>

Esta etiqueta sustituye a la etiqueta <body> y tiene tres funciones importantes:

1) Indicar al browser que la página cargada contiene frames.

2) Definir el número de columnas y renglones en los que deberá ser dividida la pantalla.

3) Establecer el tamaño o proporción de cada frame para su correcta visualización.

b) <FRAME>

Por su parte, esta etiqueta sirve para definir las propiedades de cada frame, las cuales son:

i) Scrolling = " yes | no | auto "

Con esta opción, definimos si la ventana tendrá barra de desplazamiento (por omisión esta en "auto").

ii) Noresize

Indica si el usuario podrá o no modificar el tamaño del frame.

iii) Name

Como todos los objetos, puede hacerse referencia a un frame por su nombre.

iv) Marginwidth = " # | % "

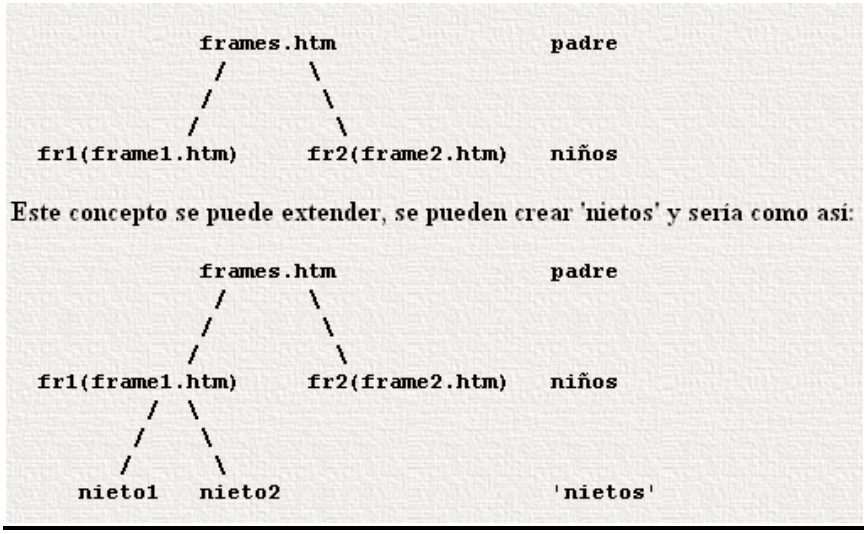
Especifica el ancho del frame, puede ser en pixeles o en porcentaje.

v) Marginheight = " # | % "

Especifica el alto del frame, puede ser en pixeles o en porcentaje.

Elementos tales como las formas, los botones, las hiperligas, etc. tienen definida la propiedad Target = "<frame>"; con la cual, podemos definir en que frame o ventana deberán ser arrojados los resultados de hacer click en cualquiera de ellos.

¿CÓMO USAR FRAMES CON JAVASCRIPT?



- **ENCABEZADOS**

El JavaScript se puede colocar dentro de las etiquetas <head>, es decir, en el encabezado de una página. De tal forma que se ejecuta al momento de cargar el sitio. Esto es muy bueno para declarar funciones y desatar el evento.

En el siguiente ejemplo la función mensaje se ejecuta cuando se carga el cuerpo de la página.

```
<html>
<head>
<script type="text/javascript">
function mensaje(){
```

```
alert("Esta es una alerta que se ejecuta al cargar el body, por el evento onload.");
}
</script>
</head>
<body onload="mensaje()">
</body>
</html>
```

- **COOKIES Y JAVASCRIPT**

Las cookies son pequeños paquetes de información enviadas por las webs que visitas y que el navegador almacena en el pc del usuario. Son útiles para guardar preferencias del usuario u otros datos referentes a usuarios en particular.

¿QUÉ INFORMACIÓN SE PUEDE ALMACENAR EN UNA COOKIE?

Se puede almacenar una alta variedad de información. Esta información se almacena en cadenas de texto en la forma variable=valor. Has de tener en cuenta que todas estas variables van en una sola cadena de texto no más de 4KB y cada par variable/valor se separa de la siguiente por un ; de la siguiente forma:

"estilo=rojo; max-age=40; path=/; domain=efxto.com"

Analizemos el código que he escrito. max-age, path y domain son nombres de variables reservados pues son leidos por el navegador para controlar información como la vida útil de la cookie o la web para la que es válida. La variable estilo es la que yo he introducido y cuyo valor va a ser el estilo seleccionado por el usario para que sea el mismo cuando vuelva a mi página. Puedes crear la variable que quieras y darle cualquier valor que desees, con las siguientes particularidades:

max-age

Por defecto toda cookie tiene una vida útil de la sesión del navegador, en otras palabras, cuando cierras el navegador la cookie desaparece. La variable max-age puede alterar este valor por defecto para hacer que sea válida durante más tiempo. Para especificar el tiempo que la cookie será válida simplemente dale el valor de tiempo que quieras a la variable max-age siempre expresado en segundos.

Por ejemplo, si queremos que la cookie sea válida por 60 días pondremos:

"estilo=rojo; max-age=51840000; path=/; domain="bloogie.es"

Actualmente la mayoría de navegadores admiten otra forma en la que no es necesario calcular los segundos: (60 segundos por 60 mintuos por 24 horas * 60 días):

"estilo=rojo; max-age=" +60*60*24*60 + ; path=/; domain="bloogie.es"

path

Toda cookie es válida por defecto sólo para las páginas en el directorio de la página actual e inferiores. Es decir, si la página http://dominio.com/seccion-

1/articulo3.html envía una cookie, esta será válida también para todas las páginas bajo http://dominio.com/seccion-1/ y para páginas inferiores como por ejemplo http://dominio.com/seccion-1/categoria-3/articulo4.html pero no para http://dominio.com o http://dominio.com/seccion-2.

A través de la variable path se puede alterar este valor por defecto. Por ejemplo si quieres que la cookie sea válida para todo tu sitio le daremos el valor "/" a la variable path, o si queremos que sea válida para un determinado directorio le daremos el valor "/directorio".

domain

Con esta variable podemos especificar sub-dominios para los que la cookie es válida, he dicho subdominios porque si tu sitio es dominio.com, el navegador no aceptará cookies para otrodominio.com o facebook.com. Lo que podemos hacer con la variable domain es decirle al navegador si la cookie es válida solo para el domino principal (domain=www.dominio.com), para el dominio principal y para los subdominios (domain=dominio.com) o para un subdominio concreto (domain=subdominio.dominio.com),

secure

Esta variable no debe ser especificada a menos que quieras que la cookie sea enviada sólo si el usuario está visitando tu web a través de una conexión segura.

expires

Esta es una variable depreciada aunque aún sea soportada por los navegadores. La variable max-age es la que reemplaza a expires. Sin embargo ten encuenta que no debes usar el nombre "expires" para almacenar tu información.

No pongas espacios, comas o semi-colons en el valor de las variables

El valor de las variables de una cookie no puede contener espacios en blanco, comas o putno y comas (semi-colons). Si inevitablemente tienes que usar estos caracteres debes codificarlos. Entre los métodos que puedes usar el más fácil es encodeURIComponent() para codificar y decodeURIComponent() para decodificar cuando leas la cookie. Por ejemplo:

"estilo=" + encondeURIComponent("estilo rojo") + "; max-age=" + 60*60*24*60 +"; path="/"; domain=bloogie.es"

LÍMITES DE USO DE COOKIES

Cada navegador puede tener implementado límites difrentes para las cookies pero los estándares mínimos que debe tener son:

Longitud de la cookie: 4KB. Todas las variables, incluyendo las especiales, no deben tener más de 4.096 caracteres.

20 cookies como máximo por web.

300 cookies almacenadas como máximo, incluyendo cookies enviadas por otros sitios web.

COMO ENVIAR UNA COOKIE CON JAVASCRIPT

Enviar una cookie es bastante simple, sólo necesitas darle a la propiedad document.cookie el valor de la cadena de texto con las variables de tu cookie:

document.cookie="estilo=" + encondeURIComponent("estilo rojo" +"; max-age=" + 60*60*24*60 +"; path=/; domain=bloogie.es";

La siguiente función facilita algo el trabajo:

```
function enviar_cookie (nombre_cookie,valor_cookie,vida_cookie_dias,dominio_cookie){
  document.cookie = nombre_cookie +
  "=" + encodeURIComponent( valor_cookie ) +
  "; max-age=" + 60 * 60 *
  24 * vida_cookie_dias +
  "; path=/; domain=" + dominio_cookie ;
}
```

Para establecer la cookie ahora solo tienes que llamar la función desde la parte de una página que quieras mediante una expresión que para nuestro ejemplo sería:

enviar_cookie("estilo","color rojo","60","bloogie.es");

COMO LEER UNA COOKIE

En este punto ya hemos enviado una cookie, pero no nos servirá de nada si no la leemos cuando el usuario visite nuestra página.

Lo primero será comprobar si la cookie existe, esto si la hemos enviado y permanece guardada en el navegador. Una vez obtenida la leeremos como lo que és, una cadena de texto en busca de los parámetros que nos interesen. En nuestro ejemplo el parámetro que nos interesa es el nombre de la cookie y su valor, es decir, que valor tiene el parámetro que hemos introducido llamado "estilo". Para hacer esto podemos usar expresiones regulares o funciones javascript como split para separar la cadena de texto en varias partes y buscar una subcadena. Para nuestro ejemplo se podría usar la siguiente función:

```
function get_cookie ( nombre_cookie ){
  if (document.cookie.length != 0) {
    var valor_cookie = docuemnt.cookie.match (
     '(^|;)[\s]*' +
      nombre_cookie +
      '=([^;]*)' );
      return decodeURIComponent ( valor_cookie[2] ) ;
   }
  return " ;
}
```

Para obtener el estilo seleccionado por el usuario previamente y establecido en la cookie que enviamos utilizaremos:

estilo_usuario = get_cookie("estilo");

COMO BORRAR UNA COOKIE

Alguna vez puede que necesites borrar una cookie enviada con anterioridad. Para hacer esto envía una cookie con max-age=0 para el mismo path y domain, dejando el parámetro que deseas eliminar con valor nulo (en nuestro caso el parámetro estilo). La función anterior que usamos para enviar la cookie dejaba el path=/, para borrar esta cookie podríamos usar esta función:

```
function borrar_cookie ( nombre_cookie, dominio_cookie ){
 document.cookie = nombre_cookie + "="; max-age=0; path=/; domain=" + dominio_cookie ;
}
```

Llamaremos a la función con:

borrar_cookie("estilo","bloogie.es");

- ***FUNCIONES Y OBJETOS***

DEFINICIÓN DE UNA FUNCIÓN

```
function nombre_funcion(argumentos) {
   bloque de comandos
}
```

El nombre de la función es sensible a mayúsculas y minúsculas. Puede incluir el caracter "_" y empezar con una letra.

Ejemplo:

```
function ImprimeNombre(nombre) {
   document.write("<HR>Tu nombre es <B><I>")
   document.write(nombre)
   document.write("</B></I><HR>")
}
```

La variable nombre sólo existe dentro de la función y lo mismo pasa para cualquier variable declarada dentro de la función.

Las funciones pueden devolver un valor. Para ello:

```
function cubo(numero) {
   var cubo = numero * numero * numero
   return cubo
}
```

También podría haber sido

```
return numero * numero * numero
```

La función eval() evalúa un string devolviendo un valor numérico. Por ejemplo, eval("10*10") devolverá el valor 100.

FUNCIONES RECURSIVAS

Son las que se llaman a sí mismas. Un ejemplo típico es el de la función que calcula el factorial de un número:

```
function factorial(numero) {
   if (numero > 1) {
      return numero * factorial(numero - 1)
   } else {
      return numero
   }
}
```

Esta función se basa en que el factorial de un número x es igual x multiplicado por el factorial de x - 1:

x! = x * (x - 1)!

CREACIÓN DE OBJETOS

Para crear objetos, primero es necesario definir su tipo (o clase):

```
function empleado(nombre, edad, puesto) {
        this.nombre = nombre
        this.edad = edad
        this.puesto = puesto
}
```

En JavaScript, la partícula this se refiere al objeto en el que se utiliza. Una vez definida la clase, podemos crear variables (instanciar objetos) de esa clase de la siguiente manera:

```
empleado_1 = new empleado("Pedro", 26, "Programador")
```

Pueden añadirse propiedades a los objetos aunque estas no haya sido declaradas en la definición de la clase. Por ejemplo:

```
empleado_1.jefe = "Luis"
```

Estas propiedades nuevas sólo afectaran a ese objeto y no al resto.

Los objetos pueden anidarse de forma que un objeto sea un a propiedad de otro objeto. Por ejemplo:

```
function oficina(ciudad, pais) {
   this.ciudad = ciudad
```

```
    this.pais = pais
}
oficinaPedro = new oficina("Madrid","España")
empleado_1 = new empleado("Pedro", 26, "Programador", oficinaPedro)
```

En el ejemplo anterior, hay que definir la clase empleado de esta forma:

```
function empleado(nombre, edad, puesto, oficina)
    this.nombre = nombre
    this.edad = edad
    this.puesto = puesto
    this.oficina = oficina
}
```

Dentro de la definición de la clase o tipo del objeto, pueden incluirse funciones que accedan a sus propiedades. Estas funciones reciben el nombre de métodos. Un método se define de la siguiente manera:

```
function mostrarPerfil() {
    document.write("Nombre: " + this.nombre + "<BR>")
    document.write("Edad: " + this.edad + "<BR>")
    document.write("Puesto: " + this.puesto + "<BR>")
}

function empleado(nombre, edad, puesto) {
this.nombre = nombre
this.edad = edad
this.puesto = puesto
this.mostrarPerfil = mostrarPerfil
}
```

ARRAYS ASOCIATIVOS

En JavaScript las propiedades de un objeto y los arrays están relacionados de la siguiente manera:

```
empleado_1[0] = "Pedro"
empleado_1[1] = 26
empleado_1[2] = "Programador"
```

Y también se produce la siguiente equivalencia:

empleado_2["nombre"] equivale a empleado_2[0] empleado_2["edad"] equivale a empleado_2[1]