



Fuente: <http://www.desarrolloweb.com/articulos/490.php?manual=20>

Introducción a Javascript

Qué es JavaScript y las posibilidades que nos ofrece con respecto al HTML
Javascript es un lenguaje de programación utilizado para crear pequeños programitas encargados de realizar acciones dentro del ámbito de una página web. Con Javascript podemos crear efectos especiales en las páginas y definir interactividades con el usuario. El navegador del cliente es el encargado de interpretar las instrucciones Javascript y ejecutarlas para realizar estos efectos e interactividades, de modo que el mayor recurso, y tal vez el único, con que cuenta este lenguaje es el propio navegador.

Javascript es el siguiente paso, después del HTML, que puede dar un programador de la web que decida mejorar sus páginas y la potencia de sus proyectos. Es un lenguaje de programación bastante sencillo y pensado para hacer las cosas con rapidez, a veces con ligereza. Incluso las personas que no tengan una experiencia previa en la programación podrán aprender este lenguaje con facilidad y utilizarlo en toda su potencia con sólo un poco de práctica.

Entre las acciones típicas que se pueden realizar en Javascript tenemos dos vertientes. Por un lado los efectos especiales sobre páginas web, para crear contenidos dinámicos y elementos de la página que tengan movimiento, cambios de color o cualquier otro dinamismo. Por el otro, javascript nos permite ejecutar instrucciones como respuesta a las acciones del usuario, con lo que podemos crear páginas interactivas con programas como calculadoras, agendas, o tablas de cálculo.

Javascript es un lenguaje con muchas posibilidades, permite la programación de pequeños scripts, pero también de programas más grandes, orientados a objetos, con funciones, estructuras de datos complejas, etc. Toda esta potencia de Javascript se pone a disposición del programador, que se convierte en el verdadero dueño y controlador de cada cosa que ocurre en la página.

En este libro vamos a tratar de acercarnos a este lenguaje en profundidad y conocer todos sus secretos y métodos de trabajo. Al final del libro seremos capaces de controlar la página web y discernir el mejor método para atacar los problemas u objetivos que nos hayamos planeado.

Algo de historia

Relatamos los orígenes y etimologías de este lenguaje.

En Internet se han creado multitud de servicios para realizar muchos tipos de comunicaciones, como correo, charlas, búsquedas de información, etc. Pero ninguno de estos servicios se ha desarrollado tanto como el Web. Si estamos leyendo estas líneas no vamos a necesitar ninguna explicación de lo que es el web, pero si podemos hablar un poco sobre cómo se ha ido desarrollando con el paso de los años.

El web es un sistema Hipertexto, una cantidad desmesurada de textos que contienen enlaces que relacionan cada una de las unidades básicas donde podemos encontrar información, las páginas web. En un principio, para diseñar este sistema de páginas con enlaces se pensó en un lenguaje que permitiese presentar cada una de estas informaciones junto con unos pequeños estilos, este lenguaje fue el HTML, que luego se vería que no cumplió todos los objetivos para los que fue diseñado, pero eso es otro tema.

El caso es que HTML no es suficiente para realizar todas las acciones que se pueden llegar a necesitar en una página web. Esto es debido a que conforme fue creciendo el web y sus distintos usos se fueron complicando las páginas y las acciones que se querían realizar a través de ellas. El HTML se había quedado corto para definir todas estas nuevas funcionalidades, ya que sólo sirve para presentar el texto en una página, definir su estilo y poco más.

El primer ayudante para cubrir las necesidades que estaban surgiendo fue Java, a través de la tecnología de los Applets, que son pequeños programas que se incrustan en las páginas web y que pueden realizar las acciones asociadas a los programas de propósito general. La programación de Applets fue un gran avance y Netscape, por aquel entonces el navegador más popular, había roto la primera barrera del HTML al hacer posible la programación dentro de las páginas web. No cabe duda que la aparición de los Applets supuso un gran avance en la historia del web, pero no ha sido una tecnología definitiva y muchas otras han seguido implementando el camino que comenzó con ellos.

Llega Javascript:

Netscape, después de hacer sus navegadores compatibles con los applets, comenzó a desarrollar un lenguaje de programación al que llamó LiveScript que permitiese crear pequeños programas en las páginas y que fuese mucho más sencillo de utilizar que Java. De modo que el primer Javascript se llamo LiveScript, pero no duró mucho ese nombre, pues antes de lanzar la primera versión del producto se forjó una alianza con Sun Microsystems, creador de Java, para desarrollar en conjunto ese nuevo lenguaje.

La alianza hizo que Javascript se diseñara como un hermano pequeño de Java, solamente útil dentro de las páginas web y mucho más fácil de utilizar, de modo que cualquier persona, sin conocimientos de programación pudiese adentrarse en el lenguaje y utilizarlo a sus anchas. Además, para programar Javascript no es necesario un kit de desarrollo, ni compilar los scripts, ni realizarlos en ficheros externos al código HTML, como ocurría con los applets.

Netscape 2.0 fue el primer navegador que entendía Javascript y su estela fue seguida por los navegadores de la compañía Microsoft a partir de la versión 3.0.

Diferencias entre Java y Javascript

Ponemos de manifiesto la diferencia entre estos dos lenguajes con un origen común.

Queremos que quede claro que **Javascript no tiene nada que ver con Java**, salvo en sus orígenes, como se ha podido leer hace unas líneas. Actualmente son productos totalmente distintos y no guardan entre si más relación que la sintaxis idéntica y poco más. Algunas diferencias entre estos dos lenguajes son las siguientes:

- **Compilador.** Para programar en Java necesitamos un Kit de desarrollo y un compilador. Sin embargo, Javascript no es un lenguaje que necesite que sus programas se compilen, sino que éstos se interpretan por parte del navegador cuando éste lee la página.
- **Orientado a objetos.** Java es un lenguaje de programación orientado a objetos. (Más tarde veremos que quiere decir orientado a objetos, para el que no lo sepa todavía) Javascript no es orientado a objetos, esto quiere decir que podremos programar sin necesidad de crear clases, tal como se realiza en los lenguajes de programación estructurada como C o Pascal.
- **Propósito.** Java es mucho más potente que Javascript, esto es debido a que Java es un lenguaje de propósito general, con el que se pueden hacer aplicaciones de lo más variado, sin embargo, con Javascript sólo podemos escribir programas para que se ejecuten en páginas web.
- **Estructuras fuertes.** Java es un lenguaje de programación fuertemente tipado, esto quiere decir que al declarar una variable tendremos que indicar su tipo y no podrá cambiar de un tipo a otro automáticamente. Por su parte Javascript no tiene esta característica, y podemos meter en una variable la información que deseemos, independientemente del tipo de ésta. Además, podremos cambiar el tipo de información de una variable cuando queramos.
- **Otras características.** Como vemos Java es mucho más complejo, aunque también más potente, robusto y seguro. Tiene más funcionalidades que Javascript y las diferencias que los separan son lo suficientemente importantes como para distinguirlos fácilmente.

Antes de empezar

Mostramos ejemplos de páginas que emplean JavaScript en su desarrollo y comentamos las aplicaciones necesarias para empezar a programar.

Previamente a comenzar a utilizar Javascript podemos hacernos una idea más concreta de las posibles aplicaciones de este lenguaje así como las herramientas que necesitamos para ponernos manos a la obra.

Usos de Javascript

Veamos brevemente algunos usos de este lenguaje que podemos encontrar en el web para hacernos una idea de las posibilidades que tiene.

Para empezar, podemos ver páginas como la página de [SEAT](#), llena de efectos súper interesantes sobre Javascript, que llegan a asemejarse a la tecnología Flash. En esta misma vertiente de uso de Javascript podemos encontrar muchas páginas, por ejemplo la página [Guia de Multimedia](#), que realiza una animación Javascript para presentar a la empresa dueña de la página web.

Por otro lado, podemos encontrar dentro de Internet muchas aplicaciones de Javascript mucho más serias, que hacen que una página web se convierta en un verdadero programa interactivo de gestión de cualquier recurso. Por ejemplo podemos ver una página que consiste en un test de preguntas, que recoge los resultados y los envía a su evaluador. Esta página fue diseñada en su día por el escritor de este libro.

La dirección es <http://www.geocities.com/Athens/Oracle/3391/>. También se pueden ver más ejemplos de estos dentro de cualquier página un poco compleja, si nos pasamos por un sitio que tenga una calculadora o un convertidor de divisas, veremos que en muchos casos se han realizado con Javascript.

En realidad es mucho más habitual encontrar Javascript para realizar efectos simples sobre páginas web, o no tan simples, como pueden ser rollovers (que cambie una imagen al pasar el ratón por encima), navegadores desplegables, apertura de ventanas secundarias, etc. Nos atrevemos a decir que este lenguaje es realmente útil para estos casos, pues estos típicos efectos tienen la complejidad justa para ser implementados en cuestión de minutos sin posibilidad de errores. Las páginas de [Desarrollo Web](#) son un ejemplo de páginas que utilizan Javascript para realizar multitud de acciones sin que estas sean demasiado complicadas, que carguen la página o que den lugar a errores en distintas plataformas.

Qué necesitas

Para programar en Javascript necesitamos básicamente lo mismo que para programar páginas web con HTML. Un editor de textos y un navegador compatible con Javascript. Un usuario de Windows posee de salida todo lo necesario para poder programar en Javascript, puesto que dispone dentro de su instalación típica de sistema operativo, de un editor de textos, el Bloc de notas, y de un navegador: Internet Explorer.

Usuarios de otros sistemas pueden encontrar en Internet fácilmente las herramientas necesarias para comenzar en páginas de descarga de software como [Tucows](#).

Permitidme una recomendación con respecto al editor de textos. Se trata de que, aunque el Bloc de Notas es suficiente para empezar, tal vez sea muy útil contar con otros programas que nos ofrecen mejores prestaciones a la hora de escribir las líneas de código. Estos editores avanzados tienen algunas ventajas como que colorean los códigos de nuestros scripts, nos permiten trabajar con varios documentos simultáneamente, tienen ayudas, etc. Entre otros queremos destacar el [Home Site](#) o [UltraEdit](#).

Versiones de navegadores y de Javascript

Presentamos las diferentes versiones de JavaScript, los navegadores que las aceptan y sus contribuciones con respecto a las predecesoras.

También resulta apropiado introducir las distintas versiones de Javascript que existen y que han evolucionado en conjunto con las versiones de navegadores. El lenguaje ha ido avanzando durante sus años de vida e incrementando sus capacidades. En un principio podía realizar muchas cosas en la página web, pero tenía pocas instrucciones para crear efectos especiales. Con el tiempo también el HTML ha avanzado y se han creado nuevas características como las capas, que permiten tratar y maquetar los documentos de manera distinta. Javascript ha avanzado también y para manejar todas estas nuevas características se han creado nuevas instrucciones y recursos. Para resumir vamos a comentar las distintas versiones de Javascript:

- **Javascript 1:** nació con el Netscape 2.0 y soportaba gran cantidad de instrucciones y funciones, casi todas las que existen ahora ya se introdujeron en el primer estándar.
- **Javascript 1.1:** Es la versión de Javascript que se diseñó con la llegada de los navegadores 3.0. Implementaba poco más que su anterior versión, como por ejemplo el tratamiento de imágenes dinámicamente y la creación de arrays.
- **Javascript 1.2:** La versión de los navegadores 4.0. Esta tiene como desventaja que es un poco distinta en plataformas Microsoft y Netscape, ya que ambos navegadores crecieron de distinto modo y estaban en plena lucha por el mercado.
- **Javascript 1.3:** Versión que implementan los navegadores 5.0. En esta versión se han limado algunas diferencias y asperezas entre los dos navegadores.
- **Javascript 1.5:** Versión actual, en el momento de escribir estas líneas, que implementa Netscape 6.
- Por su parte, **Microsoft** también ha evolucionado hasta presentar su **versión 5.5 de JScript** (así llaman al javascript utilizado por los navegadores de Microsoft).

Efectos rápidos con Javascript

Antes de meternos de lleno, veamos algunos ejemplos de códigos sencillos de gran utilidad.

Antes de meternos en materia podemos ver una serie de efectos rápidos que se pueden programar con Javascript. Esto nos puede hacer una idea más clara de las capacidades y potencia del lenguaje que nos vendrán bien para tener una idea más exacta de lo que es Javascript a la hora de recorrer los siguientes capítulos.

Abrir una ventana secundaria

Primero vamos a ver que con una línea de Javascript podemos hacer cosas bastante atractivas. Por ejemplo podemos ver cómo abrir una ventana secundaria sin barras de menús que muestre el buscador Google. El código sería el siguiente.

```
<script>
window.open("http://www.google.com", "", "width=550,height=420,menubar=no")
</script>
```

Podemos [ver el ejemplo en marcha aquí](#).

Un mensaje de bienvenida

Podemos mostrar una caja de texto emergente al terminarse de cargar la portada de nuestro sitio web, que podría dar la bienvenida a los visitantes.

```
<script>  
window.alert("Bienvenido a mi sitio web. Gracias...")  
</script>
```

Puedes [ver el ejemplo en una página a parte](#).

Fecha actual

Veamos ahora un sencillo script para mostrar la fecha de hoy. A veces es muy interesante mostrarla en las webs para dar un efecto de que la página está al "al día", es decir, está actualizada.

```
<script> document.write(new Date()) </script>
```

Estas líneas deberían introducirse dentro del cuerpo de la página en el lugar donde queramos que aparezca la fecha de última actualización. Podemos [ver el ejemplo en marcha aquí](#).

Nota: Un detalle a destacar es que la fecha aparece en un formato un poco raro, indicando también la hora y otros atributos de la misma, pero ya aprenderemos a obtener exactamente lo que deseemos en el formato correcto.

Botón de volver

Otro ejemplo rápido se puede ver a continuación. Se trata de un botón para volver hacia atrás, como el que tenemos en la barra de herramientas del navegador. Ahora veremos una línea de código que mezcla HTML y Javascript para crear este botón que muestra la página anterior en el historial, si es que la hubiera.

```
<input type=button value=Atrás onclick="history.go(-1)">
```

El botón sería parecido al siguiente. Podemos pulsarlo para ver su funcionamiento (debería llevarnos a la página anterior).

Como diferencia con los ejemplos anteriores, hay que destacar que en este caso la instrucción Javascript se encuentra dentro de un atributo de HTML, onclick, que indica que esa instrucción se tiene que ejecutar como respuesta a la pulsación del botón.

Se ha podido comprobar la facilidad con la que se pueden realizar algunas acciones interesantes, existirían muchas otras muestras que nos reservamos para capítulos posteriores.

El lenguaje Javascript

Cómo se escribe en Javascript. Las primeras reglas de insertar scripts en páginas web.

En esta parte del libro vamos a conocer la manera de trabajar con Javascript, como incluir scripts y ser compatible con todos los navegadores. Muchas ideas del funcionamiento de Javascript ya se han descrito en capítulos anteriores, pero con el objetivo de no dejarnos nada en el tintero vamos a tratar de acaparar a partir de aquí todos los datos importantes de este lenguaje.

Javascript se escribe en el documento HTML

Lo más importante y básico que podemos destacar en este momento es que la programación de Javascript se realiza dentro del propio documento HTML. Esto quiere decir que en la página se mezclan los dos lenguajes de programación, y para que estos dos lenguajes se puedan mezclar sin problemas se han de incluir unos delimitadores que separan las etiquetas HTML de las instrucciones Javascript. Estos delimitadores son las etiquetas `<SCRIPT>` y `</SCRIPT>`. Todo el código Javascript que pongamos en la página ha de ser introducido entre estas dos etiquetas.

En una misma página podemos introducir varios scripts, cada uno que podría introducirse dentro de unas etiquetas `<SCRIPT>` distintas. La colocación de estos scripts es indiferente, en un principio nos da igual donde colocarlos, pero en determinados casos esta colocación si que será muy importante. En cada caso, y llegado el momento se informará de ello convenientemente.

También se puede escribir Javascript dentro de determinados atributos de la página, como el atributo `onclick`. Estos atributos están relacionados con las acciones del usuario y se llaman manejadores de eventos.

Vamos a ver en el siguiente capítulo con más detenidamente estas dos maneras de escribir scripts, que tienen como diferencia principal el momento en que se ejecutan las sentencias.

Maneras de ejecutar scripts

Existen dos maneras de ejecutar scripts en una página, al cargar la página o como respuesta a acciones del usuario.

Existen **dos maneras de ejecutar scripts** en la página. La primera de estas maneras se trata de ejecución directa de scripts, la segunda es una ejecución como respuesta a la acción de un usuario. Veremos ahora cada una de ellas.

Ejecución directa

Es el método de ejecutar scripts más básico. En este caso se incluyen las instrucciones dentro de la etiqueta `<SCRIPT>`, tal como hemos comentado anteriormente. Cuando el navegador lee la página y encuentra un script va interpretando las líneas de código y las va ejecutando una después de otra. Llamamos a esta manera ejecución directa pues cuando se lee la página se ejecutan directamente los scripts.

Este método será el que utilicemos preferentemente en la mayoría de los ejemplos de este libro.

Respuesta a un evento

Es la otra manera de ejecutar scripts, pero antes de verla debemos hablar sobre los eventos. Los eventos son acciones que realiza el usuario. Los programas como Javascript están preparados para atrapar determinadas acciones realizadas, en este caso sobre la página, y realizar acciones como respuesta. De este modo se pueden realizar programas interactivos, ya que controlamos los movimientos del usuario y respondemos a ellos. Existen muchos tipos de eventos distintos, por ejemplo la pulsación de un botón, el movimiento del ratón o la selección de texto de la página.

Las acciones que queremos realizar como respuesta a un evento se han de indicar dentro del mismo código HTML, pero en este caso se indican en atributos HTML que se colocan dentro de la etiqueta que queremos que responda a las acciones del usuario. En el capítulo donde vimos algún ejemplo rápido ya comprobamos que si queríamos que un botón realizase acciones cuando se pulsase sobre él, debíamos indicarlas dentro del atributo onclick del botón.

Comprobamos pues que se puede introducir código Javascript dentro de determinados atributos de las etiquetas HTML. Veremos más adelante este tipo de ejecución en profundidad y los tipos de eventos que existen.

Ocultar scripts en navegadores antiguos

Cómo hacer que los scripts no molesten en navegadores que no los entienden.

Ya hemos visto que Javascript se implementó a partir de Netscape 2.0 e Internet Explorer 3.0, incluso hay navegadores que funcionan en sistemas donde sólo se puede visualizar texto y por lo tanto determinadas tecnologías, como este lenguaje, están fuera de su alcance. Así pues, no todos los navegadores del web comprenden Javascript. En los casos en los que no se interpretan los scripts, los navegadores asumen que el código de éstos es texto de la propia página web y como consecuencia, presentan los scripts en la página web como si de texto normal se tratara. Para evitar que el texto de los scripts se escriba en la página cuando los navegadores no los entienden se tienen que ocultar los con comentarios HTML (<!-- comentario HTML -->). Veamos con un ejemplo cómo se han de ocultar los scripts.

```
<SCRIPT>
<!--
Código Javascript

//-->
</SCRIPT>
```

Vemos que el inicio del comentario HTML es idéntico a cómo lo conocemos en el HTML, pero el cierre del comentario presenta una particularidad, que empieza por doble barra inclinada. Esto es debido a que el final del comentario contiene varios caracteres que Javascript reconoce como operadores y al tratar de analizarlos lanza un mensaje de error de sintaxis. Para que Javascript no lance un mensaje de error se coloca antes del comentario HTML esa doble barra, que no es más que un comentario Javascript, que conoceremos más adelante cuando hablamos de sintaxis.

El inicio del comentario HTML no es necesario comentarlo con la doble barra, dado que Javascript entiende bien que simplemente se pretende ocultar el código. Una

aclaración a este punto: si pusiésemos las dos barras en esta línea, se verían en navegadores antiguos por estar fuera de los comentarios HTML. Las etiquetas <SCRIPT> no las entienden los navegadores antiguos, por lo tanto no las interpretan, tal como hacen con cualquier etiqueta que desconocen.

<NOSCRIPT>

Existe la posibilidad de indicar un texto alternativo para los navegadores que no entienden Javascript, para informarles de que en ese lugar debería ejecutarse un script y que la página no está funcionando al 100% de sus capacidades. También podemos sugerir a los visitantes que actualicen su navegador a una versión compatible con el lenguaje. Para ello utilizamos la etiqueta <NOSCRIPT> y entre esta etiqueta y su correspondiente de cierre podemos colocar el texto alternativo al script.

```
<SCRIPT>
código
</SCRIPT>
<NOSCRIPT>
```

Este navegador no comprende los scripts que se están ejecutando, debes actualizar tu versión de navegador a una más reciente.

```
<br><br>
<a href="http://netscape.com">Netscape</a>.<br>
<a href="http://microsoft.com">Microsoft</a>.
</NOSCRIPT>
```

Más sobre colocar scripts

Últimas notas sobre cómo colocar scripts. Indicar la versión utilizada y utilizar ficheros externos.

Un par de notas adicionales sobre cómo colocar scripts en páginas web.

Lenguaje que estamos utilizando

La etiqueta <SCRIPT> tiene un atributo que sirve para indicar el lenguaje que estamos utilizando, así como la versión de este. Por ejemplo, podemos indicar que estamos programando en Javascript 1.2 o Visual Basic Script, que es otro lenguaje para programar scripts en el navegador cliente que sólo es compatible con Internet Explorer.

El atributo en cuestión es language y lo más habitual es indicar simplemente el lenguaje con el que se han programado los scripts. El lenguaje por defecto es Javascript, por lo que si no utilizamos este atributo, el navegador entenderá que el lenguaje con el que se está programando es Javascript. Un detalle donde se suele equivocar la gente sin darse cuenta es que lenguaje se escribe con dos -g- y no con -g- y con -j- como en castellano.

```
<SCRIPT LANGUAGE="javascript">
```

Ficheros externos de Javascript

Otra manera de incluir scripts en páginas web, implementada a partir de Javascript 1.1, es incluir archivos externos donde se pueden colocar muchas funciones que se

utilicen en la página. Los ficheros suelen tener extensión .js y se incluyen de esta manera.

```
<SCRIPT language=javascript src="archivo_externo.js">  
//estoy incluyendo el fichero "archivo_externo.js"  
</SCRIPT>
```

Dentro de las etiquetas <SCRIPT> se puede escribir cualquier texto y será ignorado por el navegador, sin embargo, los navegadores que no entienden el atributo SRC tendrán a este texto por instrucciones, por lo que es aconsejable poner un comentario Javascript antes de cada línea con el objetivo de que no respondan con un error.

El archivo que incluimos (en este caso archivo_externo.js) debe contener tan solo sentencias Javascript. No debemos incluir código HTML de ningún tipo, ni tan siquiera las etiquetas </SCRIPT> y </SCRIPT>.

Sintaxis Javascript

Empezamos a contar la sintaxis del lenguaje Javascript, indicando sus principales características.

El lenguaje **Javascript tiene una sintaxis muy parecida a la de Java** por estar basado en él. También es muy parecida a la del lenguaje C, de modo que si el lector conoce alguno de estos dos lenguajes se podrá manejar con facilidad con el código. De todos modos, en los siguientes capítulos vamos a describir toda la sintaxis con detenimiento, por lo que los novatos no tendrán ningún problema con ella.

Comentarios

Un comentario es una parte de código que no es interpretada por el navegador y cuya utilidad radica en facilitar la lectura al programador. El programador, a medida que desarrolla el script, va dejando frases o palabras sueltas, llamadas comentarios, que le ayudan a él o a cualquier otro a leer más fácilmente el script a la hora de modificarlo o depurarlo.

Ya se vio anteriormente algún comentario Javascript, pero ahora vamos a contarlos de nuevo. Existen dos tipos de comentarios en el lenguaje. Uno de ellos, la doble barra, sirve para comentar una línea de código. El otro comentario lo podemos utilizar para comentar varias líneas y se indica con los signos /* para empezar el comentario y */ para terminarlo. Veamos unos ejemplos.

```
<SCRIPT>  
//Este es un comentario de una línea  
/*Este comentario se puede extender  
por varias líneas.
```

```
Las que quieras*/  
</SCRIPT>
```

Mayúsculas y minúsculas

En javascript se han de respetar las mayúsculas y las minúsculas. Si nos equivocamos al utilizarlas el navegador responderá con un mensaje de error de sintaxis. Por convención los nombres de las cosas se escriben en minúsculas, salvo que se utilice un nombre con más de una palabra, pues en ese caso se escribirán con mayúsculas las iniciales de las palabras siguientes a la primera. También se puede utilizar mayúsculas en las iniciales de las primeras palabras en algunos casos, como los nombres de las clases, aunque ya veremos más adelante cuáles son estos casos y qué son las clases.

Separación de instrucciones

Las distintas instrucciones que contienen nuestros scripts se han de separar convenientemente para que el navegador no indique los correspondientes errores de sintaxis. Javascript tiene dos maneras de separar instrucciones. La primera es a través del carácter punto y coma (;) y la segunda es a través de un salto de línea.

Por esta razón Las sentencias Javascript no necesitan acabar en punto y coma a no ser que coloquemos dos instrucciones en la misma línea.

No es una mala idea, de todos modos, acostumbrarse a utilizar el punto y coma después de cada instrucción pues otros lenguajes como Java o C obligan a utilizarlas y nos estaremos acostumbrando a realizar una sintaxis más parecida a la habitual en entornos de programación avanzados.

Variables Javascript

Vemos lo que es una variable, para qué sirve y como utilizarlas en Javascript.

Una variable es un espacio en memoria donde se almacena un dato, un espacio donde podemos guardar cualquier tipo de información que necesitemos para realizar las acciones de nuestros programas. Por ejemplo, si nuestro programa realiza sumas, será muy normal que guardemos en variables los distintos sumandos que participan en la operación y el resultado de la suma. El efecto sería algo parecido a esto.

```
sumando1 = 23  
sumando2 = 33  
suma = sumando1 + sumando2
```

En este ejemplo tenemos tres variables, sumando1, sumando2 y suma, donde guardamos el resultado. Vemos que su uso para nosotros es como si tuviésemos un apartado donde guardar un dato y que se pueden acceder a ellos con sólo poner su nombre.

Los nombres de las variables han de construirse con caracteres alfanuméricos y el carácter subrayado (_). Aparte de esta, hay una serie de reglas adicionales para construir nombres para variables. La más importante es que tienen que comenzar por un carácter alfabético o el subrayado. No podemos utilizar caracteres raros como el signo +, un espacio o un \$. Nombres admitidos para las variables podrían ser

```
Edad  
paisDeNacimiento  
_nombre
```

También hay que evitar utilizar nombres reservados como variables, por ejemplo no podremos llamar a nuestra variable palabras como return o for, que ya veremos que son utilizadas para estructuras del propio lenguaje. Veamos ahora algunos nombres de variables que no está permitido utilizar

```
12meses  
tu nombre  
return  
pe%pe
```

Declaración de variables

Declarar variables consiste en definir y de paso informar al sistema de que vas a utilizar una variable. Es una costumbre habitual en los lenguajes de programación el definir las variables que se van a usar en los programas y para ello, se siguen unas reglas estrictas. Pero javascript se salta muchas reglas por ser un lenguaje un tanto libre a la hora de programar y uno de los casos en los que otorga un poco de libertad es a la hora de declarar las variables, ya que no estamos obligados a hacerlo, al contrario de lo que pasa en la mayoría de los lenguajes de programación.

De todos modos, es aconsejable declarar las variables, además de una buena costumbre y para ello Javascript cuenta con la palabra var. Como es lógico, se utiliza esa palabra para definir la variable antes de utilizarla.

```
var operando1  
var operando2
```

También se puede asignar un valor a la variable cuando se está declarando

```
var operando1 = 23  
var operando2 = 33
```

También se permite declarar varias variables en la misma línea, siempre que se separen por comas.

```
var operando1,operando2
```

Ambito de las variables

Que son las variables locales y globales y como se trabaja con ellas en Javascript.

Se le llama ámbito de las variables al lugar donde estas están disponibles. Por lo general, cuando declaramos una variable hacemos que esté disponible en el lugar donde se ha declarado, esto ocurre en todos los lenguajes de programación y como javascript se define dentro de una página web, **las variables que declaremos en la página estarán accesibles dentro de ella**. De este modo, no podremos acceder a variables que hayan sido definidas en otra página. Este es el ámbito más habitual de una variable y le llamaremos a este tipo de variables globales a la página, aunque no será el único, ya que también podemos declarar variables en lugares más acotados.

Variables globales

Como hemos dicho, las variables globales son las que están declaradas en el ámbito más amplio posible, que en Javascript es una página web. Para declarar una variable global a la página simplemente lo haremos en un script, con la palabra *var*.

```
<SCRIPT>  
var variableGlobal  
</SCRIPT>
```

Las variables globales son accesibles desde cualquier lugar de la página, es decir, desde el script donde se han declarado y todos los demás scripts de la página, incluidos los manejadores de eventos, como el onclick, que ya vimos que se podía incluir dentro de determinadas etiquetas HTML.

Variables locales

También podremos declarar variables en lugares más acotados, como por ejemplo una función. A estas variables les llamaremos locales. Cuando se declaren variables locales sólo podremos acceder a ellas dentro del lugar donde se ha declarado, es decir, si la habíamos declarado en una función solo podremos acceder a ella cuando estemos en esa función.

Las variables pueden ser locales a una función, pero también pueden ser locales a otros ámbitos, como por ejemplo un bucle. En general, son ámbitos locales cualquier lugar acotado por llaves.

```
<SCRIPT>
```

```
function miFuncion (){
    var variableLocal
}
</SCRIPT>
```

En el script anterior hemos declarado una variable dentro de una función, por lo que esa variable sólo tendrá validez dentro de la función. Se pueden ver cómo se utilizan las llaves para acotar el lugar donde está definida esa función o su ámbito.

No hay problema en declarar una variable local con el mismo nombre que una global, en este caso la variable global será visible desde toda la página, excepto en el ámbito donde está declarada la variable local ya que en este sitio ese nombre de variable está ocupado por la local y es ella quien tiene validez.

```
<SCRIPT>
var numero = 2
function miFuncion (){
    var numero = 19
    document.write(numero) //imprime 19
}
document.write(numero) //imprime 2
</SCRIPT>
```

Un consejo para los principiantes podría ser no declarar variables con los mismos nombres, para que nunca haya lugar a confusión sobre qué variable es la que tiene validez en cada momento.

Diferencias entre utilizar var o no

Como hemos dicho, en Javascript tenemos libertad para declarar o no las variables con la palabra var, pero los efectos que conseguiremos en cada caso serán distintos. En concreto, cuando utilizamos var estamos haciendo que la variable que estamos declarando sea local al ámbito donde se declara. Por otro lado, si no utilizamos la palabra var para declarar una variable, ésta será global a toda la página, sea cual sea el ámbito en el que haya sido declarada.

En el caso de una variable declarada en la página web, fuera de una función o cualquier otro ámbito más reducido, nos es indiferente si se declara o no con var, desde un punto de vista funcional. Esto es debido a que cualquier variable declarada fuera de un ámbito es global a toda la página. La diferencia se puede apreciar en una función por ejemplo, ya que si utilizamos var la variable será local a la función y si no lo utilizamos, la variable será global a la página. Esta diferencia es fundamental a la hora de controlar correctamente el uso de las variables en la página, ya que si no lo hacemos en una función podríamos sobrescribir el valor de una variable, perdiendo el dato que pudiera contener previamente.

```
<SCRIPT>
var numero = 2
```

```

function miFuncion (){
    numero = 19
    document.write(numero) //imprime 19
}
document.write(numero) //imprime 2
//llamamos a la función
miFuncion()
document.write(numero) //imprime 19
</SCRIPT>

```

En este ejemplo, tenemos una variable global a la página llamada número, que contiene un 2. También tenemos una función que utiliza la variable numero sin haberla declarado con var, por lo que la variable numero de la función será la misma variable global numero declarada fuera de la función. En una situación como esta, al ejecutar la función se sobrescribirá la variable numero y el dato que había antes de ejecutar la función se perderá.

Qué podemos guardar en variables

Vemos el concepto de tipos de datos para el lenguaje Javascript y por qué es importante manejarlos bien.

En una variable podemos introducir varios tipos de información, por ejemplo texto, números enteros o reales, etc. A estas distintas clases de información se les conoce como tipos de datos. Cada uno tiene características y usos distintos, veamos cuáles son los tipos de datos de Javascript.

Números

Para empezar tenemos el tipo numérico, para guardar números como 9 o 23.6

Cadenas

El tipo cadena de carácter guarda un texto. Siempre que escribamos una cadena de caracteres debemos utilizar las comillas ("").

Booleanos

También contamos con el tipo booleano, que guarda una información que puede valer si (true) o no (false).

Por último sería relevante señalar aquí que nuestras variables pueden contener cosas más complicadas, como podría ser un objeto, una función, o vacío (null) pero ya lo veremos más adelante.

En realidad nuestras variables no están forzadas a guardar un tipo de datos en concreto y por lo tanto no especificamos ningún tipo de datos para una variable cuando la estamos declarando. Podemos introducir cualquier información en una variable de cualquier tipo, incluso podemos ir cambiando el contenido de una variable de un tipo a otro sin ningún problema. Vamos a ver esto con un ejemplo.

```

var nombre_ciudad = "Valencia"
var revisado = true
nombre_ciudad = 32
revisado = "no"

```

Esta ligereza a la hora de asignar tipos a las variables puede ser una ventaja en un principio, sobretodo para personas inexpertas, pero a la larga puede ser fuente de errores ya que dependiendo del tipo que son las variables se comportarán de un modo u otro y si no controlamos con exactitud el tipo de las variables podemos encontrarnos sumando un texto a un número. Javascript operará perfectamente, y devolverá un dato, pero en algunos casos puede que no sea lo que estábamos esperando. Así pues, aunque tenemos libertad con los tipos, esta misma libertad nos hace estar más atentos a posibles desajustes difíciles de detectar a lo largo de los programas. Veamos lo que ocurriría en caso de sumar letras y números.

```
var sumando1 = 23  
var sumando2 = "33"  
var suma = sumando1 + sumando2  
document.write(suma)
```

Este script nos mostraría en la página el texto 2333, que no se corresponde con la suma de los dos números, sino con su concatenación, uno detrás del otro.

Veremos algunas cosas más referentes a los tipos de datos más adelante.

Tipos de datos en Javascript

Vemos los tres tipos de datos que soporta Javascript: numérico, booleano y texto.

En nuestros scripts vamos a manejar variables diversas clases de información, como textos o números. Cada una de estas clases de información son los tipos de datos. Javascript distingue entre tres tipos de datos y todas las informaciones que se puedan guardar en variables van a estar encajadas en uno de estos tipos de datos. Veamos detenidamente cuáles son estos tres tipos de datos.

Tipo de datos numérico

En este lenguaje sólo existe un tipo de datos numérico, al contrario que ocurre en la mayoría de los lenguajes más conocidos. Todos los números son por tanto del tipo numérico, independientemente de la precisión que tengan o si son números reales o enteros. Los números enteros son números que no tienen coma, como 3 o 339. Los números reales son números fraccionarios, como 2.69 o 0.25, que también se pueden escribir en notación científica, por ejemplo 2.482e12.

Con Javascript también podemos escribir números en otras bases, como la hexadecimal. Las bases son sistemas de numeración que utilizan más o menos dígitos para escribir los números. Existen tres bases con las que podemos trabajar

- Base 10, es el sistema que utilizamos habitualmente, el sistema decimal. Cualquier número, por defecto, se entiende que está escrito en base 10.

- Base 8, también llamado sistema octal, que utiliza dígitos del 0 al 7. Para escribir un número en octal basta con escribir ese número precedido de un 0, por ejemplo 045.
- Base 16 o sistema hexadecimal, es el sistema de numeración que utiliza 16 dígitos, los comprendidos entre el 0 y el 9 y las letras de la A a la F, para los dígitos que faltan. Para escribir un número en hexadecimal debemos escribirlo precedido de un cero y una equis, por ejemplo 0x3EF.

Tipo booleano

El tipo boolean, boolean en inglés, sirve para guardar un si o un no o dicho de otro modo, un verdadero o un falso. Se utiliza para realizar operaciones lógicas, generalmente para realizar acciones si el contenido de una variable es verdadero o falso.

Si una variable es verdadero entonces Ejecuto unas instrucciones Si no Ejecuto otras

Los dos valores que pueden tener las variables booleanas son true o false.

```
miBoleana = true
miBoleana = false
```

Tipo de datos cadena de caracteres

El último tipo de datos es el que sirve para guardar un texto. Javascript sólo tiene un tipo de datos para guardar texto y en él se pueden introducir cualquier número de caracteres. Un texto puede estar compuesto de números, letras y cualquier otro tipo de caracteres y signos. Los textos se escriben entre comillas, dobles o simples.

```
miTexto = "Pepe se va a pescar"
miTexto = '23%$ Letras & *--*' 
```

Todo lo que se coloca entre comillas, como en los ejemplos anteriores es tratado como una cadena de caracteres independientemente de lo que coloquemos en el interior de las comillas. Por ejemplo, en una variable de texto podemos guardar números y en ese caso tenemos que tener en cuenta que las variables de tipo texto y las numéricas no son la misma cosa y mientras que las de numéricas nos sirven para hacer cálculos matemáticos las de texto no.

Caracteres de escape en cadenas de texto.

Hay una serie de caracteres especiales que sirven para expresar en una cadena de texto determinados controles como puede ser un salto de línea o un tabulador. Estos son los caracteres de escape y se escriben con una notación especial que comienza por una contra barra (una barra inclinada al revés de la normal '\') y luego se coloca el código del carácter a mostrar.

Un carácter muy común es el salto de línea, que se consigue escribiendo \n. Otro carácter muy habitual es colocar unas comillas, pues si colocamos unas comillas sin su carácter especial nos cerrarán las comillas que colocamos para iniciar la cadena de caracteres. Las comillas las tenemos que introducir entonces con \" o \' (comillas dobles o simples). Existen otros caracteres de escape, que veremos en la tabla de

abajo más resumidos, aunque también hay que destacar como carácter habitual el que se utiliza para escribir una contrabarra, para no confundirla con el inicio de un carácter de escape, que es la doble contrabarra \\.

Tabla con todos los caracteres de escape

Salto de línea: \n
Comilla simple: \'
Comilla doble: \"
Tabulador: \t
Retorno de carro: \r
Avance de página: \f
Retroceder espacio: \b
Contrabarra: \\

Algunos de estos caracteres probablemente no los llegarás a utilizar nunca, pues su función es un poco rara y a veces poco clara.

Operadores Javascript I

Estudiamos lo que es un operador y para qué sirve. Vemos los operadores aritméticos y de asignación.

Al desarrollar programas en cualquier lenguaje se utilizan los operadores. Éstos sirven para hacer los cálculos y operaciones necesarios para llevar a cabo sus objetivos. Un programa que no realiza operaciones solo se puede limitar a hacer siempre lo mismo, es el resultado de estas operaciones lo que hace que un programa varíe su comportamiento según los datos que obtenga. Existen operaciones más sencillas o complejas, que se pueden realizar con operandos de distintos tipos de datos, como números o textos, veremos en este capítulo de manera detallada todos estos operadores.

Ejemplos de uso de operadores

Antes de entrar a enumerar los distintos tipos de operadores vamos a ver un par de ejemplos de éstos para que nos ayuden a hacernos una idea más exacta de lo que son. En el primer ejemplo vamos a realizar una suma utilizando el operador suma.

3 + 5

Esta es una expresión muy básica que no tiene mucho sentido ella sola. Hace la suma entre los dos operandos número 3 y 5, pero no sirve de mucho porque no se hace nada con el resultado. Normalmente se combinan más de un operador para crear expresiones más útiles. La expresión siguiente es una combinación entre dos operadores, uno realiza una operación matemática y el otro sirve para guardar el resultado.

```
miVariable = 23 * 5
```

En el ejemplo anterior, el operador `*` se utiliza para realizar una multiplicación y el operador `=` se utiliza para asignar el resultado en una variable, de modo que guardemos el valor para su posterior uso.

Los operadores se pueden clasificar según el tipo de acciones que realizan. A continuación vamos a ver cada uno de estos grupos de operadores y describiremos la función de cada uno.

Operadores aritméticos

Son los utilizados para la realización de operaciones matemáticas simples como la suma, resta o multiplicación. En javascript son los siguientes:

+ Suma de dos valores

- Resta de dos valores, también puede utilizarse para cambiar el signo de un número si lo utilizamos con un solo operando `-23`

* Multiplicación de dos valores

/ División de dos valores

% El resto de la división de dos números (`3%2` devolvería 1, el resto de dividir 3 entre 2)

`++` Incremento en una unidad, se utiliza con un solo operando
`--` Decremento en una unidad, utilizado con un solo operando

Ejemplos

```
precio = 128 //introduzco un 128 en la variable precio  
unidades = 10 //otra asignación, luego veremos operadores de asignación  
factura = precio * unidades //multiplico precio por unidades, obtengo el valor factura
```

```
resto = factura % 3 //obtengo el resto de dividir la variable factura por 3  
precio++ //incrementa en una unidad el precio (ahora vale 129)
```

Operadores de asignación

Sirven para asignar valores a las variables, ya hemos utilizado en ejemplos anteriores el operador de asignación `=`, pero hay otros operadores de este tipo, que provienen del lenguaje C y que muchos de los lectores ya conocerán.

= Asignación. Asigna la parte de la derecha del igual a la parte de la izquierda.
A al derecha se colocan los valores finales y a la izquierda generalmente se coloca una variable donde queremos guardar el dato.

+= Asignación con suma. Realiza la suma de la parte de la derecha con la de la izquierda y guarda el resultado en la parte de la izquierda.

-= Asignación con resta

*= Asignación de la multiplicación

/= Asignación de la división

%= Se obtiene el resto y se asigna

Ejemplos

```
ahorros = 7000 //asigna un 7000 a la variable ahorros  
ahorros += 3500 //incrementa en 3500 la variable ahorros, ahora vale 10500  
ahorros /= 2 //divide entre 2 mis ahorros, ahora quedan 5250
```

Operadores Javascript II

Estudiamos los operadores de cadenas, lógicos y condicionales.

Operadores de cadenas

Las cadenas de caracteres, o variables de texto, también tienen sus propios operadores para realizar acciones típicas sobre cadenas. Aunque javascript sólo tiene un operador para cadenas se pueden realizar otras acciones con una serie de funciones predefinidas en el lenguaje que veremos más adelante.

+ Concatena dos cadenas, pega la segunda cadena a continuación de la primera.

Ejemplo

```
cadena1 = "holo"  
cadena2 = "mundo"  
cadenaConcatenada = cadena1 + cadena2 //cadena concatenada vale "holamundo"
```

Un detalle importante que se puede ver en este caso es que el operador + sirve para dos usos distintos, si sus operandos son números los suma, pero si se trata de cadenas las concatena. Esto pasa en general con todos los operadores que se repiten en el lenguaje, javascript es suficientemente listo para entender que tipo de operación realizar mediante una comprobación de los tipos que están implicados en ella.

Un caso que resultaría confuso es el uso del operador + cuando se realiza la operación con operadores texto y numéricos entremezclados. En este caso javascript asume que se desea realizar una concatenación y trata a los dos operandos como si de cadenas de caracteres se trataran, incluso si la cadena de texto que tenemos fuese un número. Esto lo veremos más fácilmente con el siguiente ejemplo.

```

miNumero = 23
miCadena1 = "pepe"
miCadena2 = "456"
resultado1 = miNumero + miCadena1 //resultado1 vale "23pepe"
resultado2 = miNumero + miCadena2 //resultado2 vale "23456"
miCadena2 += miNumero //miCadena2 ahora vale "45623"

```

Como hemos podido ver, también en el caso del operador `+=`, si estamos tratando con cadenas de texto y números entremezclados, tratará a los dos operadores como si fuesen cadenas.

Operadores lógicos

Estos operadores sirven para realizar operaciones lógicas, que son aquellas que dan como resultado un verdadero o un falso, y se utilizan para tomar decisiones en nuestros scripts. En vez de trabajar con números, para realizar este tipo de operaciones se utilizan operandos booleanos, que conocimos anteriormente, que son el verdadero (true) y el falso (false). Los operadores lógicos relacionan los operandos booleanos para dar como resultado otro operando booleano, tal como podemos ver en el siguiente ejemplo.

Si tengo hambre y tengo comida entonces me pongo a comer

Nuestro programa javascript utilizaría en este ejemplo un operando booleano para tomar una decisión. Primero mirará si tengo hambre, si es cierto (true) mirará si dispongo de comida. Si son los dos ciertos, se puede poner a comer. En caso de que no tenga comida o que no tenga hambre no comería, al igual que si no tengo hambre ni comida. El operando en cuestión es el operando Y, que valdrá verdadero (true) en caso de que los dos operandos valgan verdadero.

`!` Operador NO o negación. Si era true pasa a false y viceversa.

`&&` Operador Y, si son los dos verdaderos vale verdadero.

`||` Operador O, vale verdadero si por lo menos uno de ellos es verdadero.

Ejemplo

```

miBooleano = true
miBooleano = !miBooleano //miBooleano ahora vale false
tengoHambre = true
tengoComida = true
comoComida = tengoHambre && tengoComida

```

Operadores condicionales

Sirven para realizar expresiones condicionales todo lo complejas que deseemos. Estas expresiones se utilizan para tomar decisiones en función de la comparación de varios elementos, por ejemplo si un número es mayor que otro o si son iguales. Los operadores condicionales se utilizan en las expresiones condicionales para tomas de decisiones. Como estas expresiones condicionales serán objeto de estudio

más adelante será mejor describir los operadores condicionales más adelante. De todos modos aquí podemos ver la tabla de operadores condicionales.

`==` Comprueba si dos números son iguales
`!=` Comprueba si dos números son distintos
`>` Mayor que, devuelve true si el primer operador es mayor que el segundo
`<` Menor que, es true cuando el elemento de la izquierda es menor que el de la derecha
`>=` Mayor igual.
`<=` Menor igual

Operadores Javascript III

Vemos el último tipo de operador, a nivel de bit, y la precedencia de operadores (cuáles se ejecutan antes o después).

Operadores a nivel de bit

Estos son muy poco corrientes y es posible que nunca los llegues a utilizar. Su uso se realiza para efectuar operaciones con ceros y unos. Todo lo que maneja un ordenador son ceros y unos, aunque nosotros utilicemos números y letras para nuestras variables en realidad estos valores están escritos internamente en forma de ceros y unos. En algunos casos podremos necesitar realizar operaciones tratando las variables como ceros y unos y para ello utilizaremos estos operandos. En este manual se nos queda un poco grande realizar una discusión sobre este tipo de operadores, pero aquí podréis ver estos operadores por si algún día os hacen falta.

`&` Y de bits
`^` Xor de bits
`|` O de bits
`<< >> >>> >>>= >>= <<=` Varias clases de cambios

Precedencia de los operadores

La evaluación de una sentencia de las que hemos visto en los ejemplos anteriores es bastante sencilla y fácil de interpretar, pero cuando en una sentencia entran en juego multitud de operadores distintos puede haber una confusión a la hora de interpretarla y dilucidar qué operadores son los que se ejecutan antes que otros. Para marcar unas pautas en la evaluación de las sentencias y que estas se ejecuten siempre igual y con sentido común existe la precedencia de operadores, que no es más que el orden por el que se irán ejecutando las operaciones que ellos representan. En un principio todos los operadores se evalúan de izquierda a derecha, pero existen unas normas adicionales, por las que determinados operadores se evalúan antes que otros. Muchas de estas reglas de precedencia están sacadas de las matemáticas y son comunes a otros lenguajes, las podemos ver a continuación.

() [] . Paréntesis, corchetes y el operador punto que sirve para los objetos
 ! - ++ -- negación, negativo e incrementos
 * / % Multiplicación división y módulo
 +- Suma y resta
 << >> Cambios a nivel de bit
 < <= > >= Operadores condicionales
 == != Operadores condicionales de igualdad y desigualdad
 & ^ | Lógicos a nivel de bit
 && || Lógicos booleanos
 = += -= *= /= %= <<= >>= >>>= &= ^= != Asignación

En los siguientes ejemplos podemos ver cómo las expresiones podrían llegar a ser confusas, pero con la tabla de precedencia de operadores podremos entender sin errores cuál es el orden por el que se ejecutan.

$12 * 3 + 4 - 8 / 2 \% 3$

En este caso primero se ejecutan los operadores * / y %, de izquierda a derecha, con lo que se realizarían estas operaciones. Primero la multiplicación y luego la división por estar más a la izquierda del módulo.

$36 + 4 - 4 \% 3$

Ahora el módulo.

$36 + 4 - 1$

Por último las sumas y las restas de izquierda a derecha.

$40 - 1$

39

De todos modos, es importante darse cuenta que el uso de los paréntesis puede ahorrarnos muchos quebraderos de cabeza y sobretodo la necesidad de sabernos de memoria la tabla de precedencia de los operadores. Cuando veamos poco claro

el orden con el que se ejecutarán las sentencias podemos utilizarlos y así forzar que se evalúe antes el trozo de expresión que se encuentra dentro de los paréntesis.

Control de tipos

Es importante que conozcamos el tipo de las variables para trabajar sin errores. Vemos cómo obtenerlo con Javascript.

Hemos visto para determinados operadores que es importante el tipo de datos que están manejando, puesto que si los datos son de un tipo se realizarán operaciones distintas que si son de otro.

Así, cuando utilizábamos el operador +, si se trataba de números los sumaba, pero si se trataba de cadenas de caracteres los concatenaba. Vemos pues que el tipo de los datos que estamos utilizando sí que importa y que tendremos que estar pendientes este detalle si queremos que nuestras operaciones se realicen tal como esperábamos.

Para comprobar el tipo de un dato se puede utilizar otro operador que está disponible a partir de javascript 1.1, el operador typeof, que devuelve una cadena de texto que describe el tipo del operador que estamos comprobando.

```
var booleano = true
var numerico = 22
var numerico_flotante = 13.56
var texto = "mi texto"
var fecha = new Date()
document.write("<br>El tipo de booleano es: " + typeof booleano)
document.write("<br>El tipo de numerico es: " + typeof numerico)
document.write("<br>El tipo de numerico_flotante es: " + typeof
numerico_flotante)
document.write("<br>El tipo de texto es: " + typeof texto)
document.write("<br>El tipo de fecha es: " + typeof fecha)
```

Este script dará como resultado lo siguiente:

```
El tipo de booleano es: boolean
El tipo de numerico es: number
El tipo de numerico_flotante es: number
El tipo de texto es: string
El tipo de fecha es: object
```

En este ejemplo podemos ver que ser imprimen en la página los distintos tipos de las variables. Estos pueden ser los siguientes:

- boolean, para los datos booleanos. (True o false)
- number, para los numéricos.
- string, para las cadenas de caracteres.

- object, para los objetos.

Queremos destacar tan sólo dos detalles más:

- 1) Los números, ya tengan o no parte decimal, son siempre del tipo de datos numérico.
- 2) Una de las variables es un poco más compleja, es la variable fecha que es un objeto de la clase Date(), que se utiliza para el manejo de fechas en los scripts. La veremos más adelante, así como los objetos.

Estructuras de control

Introducción a las estructuras de control. Enumeramos las que tenemos disponibles en Javascript.

Los scripts vistos hasta ahora han sido tremadamente sencillos y lineales: se iban ejecutando las sentencias simples una detrás de la otra desde el principio hasta el fin. Sin embargo, esto no tiene porque ser siempre así, en los programas generalmente necesitaremos hacer cosas distintas dependiendo del estado de nuestras variables o realizar un mismo proceso muchas veces sin escribir la misma línea de código una y otra vez.

Para realizar cosas más complejas en nuestros scripts se utilizan las estructuras de control. Utilizándolas podemos realizar tomas de decisiones y bucles. En los siguientes capítulos vamos a conocer las distintas estructuras de control que existen en Javascript.

Toma de decisiones

Nos sirven para realizar unas acciones u otras en función del estado de las variables. Es decir, tomar decisiones para ejecutar unas instrucciones u otras dependiendo de lo que esté ocurriendo en ese instante en nuestros programas.

Por ejemplo, dependiendo si el usuario que entra en nuestra página es mayor de edad o no lo es, podemos permitirle o no ver los contenidos de nuestra página.

```

Si edad es mayor que 18 entonces
    Te dejo ver el contenido para adultos
Si no
    Te mando fuera de la página
  
```

En javascript podemos tomar decisiones utilizando dos enunciados distintos.

- IF
- SWITCH

Bucles

Los bucles se utilizan para realizar ciertas acciones repetidamente. Son muy utilizados a todos los niveles en la programación. Con un bucle podemos por ejemplo imprimir en una página los números del 1 al 100 sin necesidad de escribir cien veces el la instrucción imprimir.

Desde el 1 hasta el 100
Imprimir el número actual

En javascript existen varios tipos de bucles, cada uno está indicado para un tipo de iteración distinto y son los siguientes:

- FOR
- WHILE
- DO WHILE

Como hemos señalado ya, las estructuras de control son muy importantes en Javascript y en cualquier lenguaje de programación. Es por ello que en los siguientes capítulos veremos cada una de estas estructuras detenidamente, describiendo su uso y ofreciendo algunos ejemplos.

Estructura IF

Vemos cómo trabajar con la estructura de control IF en Javascript.

IF es una estructura de control utilizada para tomar decisiones. Es un condicional que realiza unas u otras operaciones en función de una expresión. Funciona de la siguiente manera, primero se evalúa una expresión, si da resultado positivo se realizan las acciones relacionadas con el caso positivo.

La sintaxis de la estructura IF es la siguiente.

```
if (expresión) {  
    acciones a realizar en caso positivo  
    ...  
}
```

Opcionalmente se pueden indicar acciones a realizar en caso de que la evaluación de la sentencia de resultados negativos.

```
if (expresión) {  
    acciones a realizar en caso positivo  
    ...  
} else {  
    acciones a realizar en caso negativo  
    ...  
}
```

Fijémonos en varias cosas. Para empezar vemos como con unas llaves engloban las acciones que queremos realizar en caso de que se cumplan o no las expresiones. Estas llaves han de colocarse siempre, excepto en el caso de que sólo haya una instrucción como acciones a realizar, que son opcionales.

Otro detalle que salta a la vista es el sangrado (margen) que hemos colocado en cada uno de los bloques de instrucciones a ejecutar en los casos positivos y negativos. Este sangrado es totalmente opcional, sólo lo hemos hecho así para que la estructura IF se comprenda de una manera más visual. Los saltos de línea tampoco son necesarios y se han colocado también para que se vea mejor la estructura. Perfectamente podríamos colocar toda la instrucción IF en la misma línea de código, pero eso no ayudará a que las cosas estén claras. Nosotros, y cualquier programador, te aconsejamos que utilices los sangrados y saltos de línea necesarios para que las instrucciones se puedan entender mejor, hoy y dentro de un mes, cuando te acuerdes menos de lo que hiciste en tus scripts.

Veamos algún ejemplo de condicionales IF.

```
if (dia == "lunes")
    document.write ("Que tengas un feliz comienzo de semana")
```

Si es lunes nos deseará una feliz semana. No hará nada en caso contrario. Como en este ejemplo sólo indicamos una instrucción para el caso positivo, no hará falta utilizar las llaves. Fíjate también en el operador condicional que consta de dos signos igual.

Vamos a ver ahora otro ejemplo, un poco más largo.

```
if (credito >= precio) {
    document.write("has comprado el artículo " + nuevoArtículo) //enseño compra
    carrito += nuevoArtículo //introduzco el artículo en el carrito de la compra
    credito -= precio //disminuyo el crédito según el precio del artículo
} else {
    document.write("se te ha acabado el crédito") //informo que te falta dinero
    window.location = "carritodelacompra.html" //voy a la página del carrito
}
```

Este ejemplo es un poco más complejo, y también un poco ficticio. Lo que hago es comprobar si tengo crédito para realizar una supuesta compra. Para ello miro si el crédito es mayor o igual que el precio del artículo, si es así informo de la compra, introduzco el artículo en el carrito y resto el precio al crédito acumulado. Si el precio del artículo es superior al dinero disponible informo de la situación y mando al navegador a la página donde se muestra su carrito de la compra.

Expresiones condicionales

La expresión a evaluar se coloca siempre entre paréntesis y está compuesta por variables que se combinan entre si mediante operadores condicionales. Recordamos que los operadores condicionales relacionaban dos variables y devolvían siempre un

resultado booleano. Por ejemplo un operador condicional es el operador "es igual" (==), que devuelve true en caso de que los dos operandos sean iguales o false en caso de que sean distintos.

```
if (edad > 18)
    document.write("puedes ver esta página para adultos")
```

En este ejemplo utilizamos en operador condicional "es mayor" (>). En este caso, devuelve true si la variable edad es mayor que 18, con lo que se ejecutaría la línea siguiente que nos informa de que se puede ver el contenido para adultos.

Las expresiones condicionales se pueden combinar con las expresiones lógicas para crear expresiones más complejas. Recordamos que las expresiones lógicas son las que tienen como operandos a los booleanos y que devuelvan otro valor booleano. Son los operadores negación lógica, Y lógico y O lógico.

```
if (bateria == 0 && redElectrica = 0)
    document.write("tu ordenador portatil se va a apagar en segundos")
```

Lo que hacemos es comprobar si la batería de nuestro supuesto ordenador está a cero (acabada) y también comprobamos si el ordenador no tiene red eléctrica (está desenchufado). Luego, el operador lógico los relaciona con un Y, de modo que si está sin batería Y sin red eléctrica, informo que el ordenador se va a apagar.

La lista de operadores que se pueden utilizar con las estructuras IF se pueden ver en el capítulo de operadores condicionales y operadores lógicos.

Estructura IF (parte II)

Vemos más cosas sobre la estructura IF: la anidación de IFs y el operador ?, un IF sencillo.

Sentencias IF anidadas

Para hacer estructuras condicionales más complejas podemos anidar sentencias IF, es decir, colocar estructuras IF dentro de otras estructuras IF. Con un solo IF podemos evaluar y realizar una acción u otra según dos posibilidades, pero si tenemos más posibilidades que evaluar debemos anidar Ifs para crear el flujo de código necesario para decidir correctamente.

Por ejemplo, si deseo comprobar si un número es mayor menor o igual que otro, tengo que evaluar tres posibilidades distintas. Primero puedo comprobar si los dos números son iguales, si lo son, ya he resuelto el problema, pero si no son iguales todavía tendré que ver cuál de los dos es mayor. Veamos este ejemplo en código Javascript.

```

var numero1=23
var numero2=63
if (numero1 == numero2){
    document.write("Los dos números son iguales")
} else {
    if (numero1 > numero2) {
        document.write("El primer número es mayor que el segundo")
    } else{
        document.write("El primer número es menor que el segundo")
    }
}

```

El flujo del programa es como comentábamos antes, primero se evalúa si los dos números son iguales. En caso positivo se muestra un mensaje informándolo. En caso contrario ya sabemos que son distintos, pero aun debemos averiguar cuál de los dos es mayor. Para eso se hace otra comparación para saber si el primero es mayor que el segundo. Si esta comparación da resultados positivos mostramos un mensaje diciendo que el primero es mayor que el segundo, en caso contrario indicaremos que el primero es menor que el segundo.

Volvemos a remarcar que las llaves son en este caso opcionales, pues sólo se ejecuta una sentencia para cada caso. Además, los saltos de línea y los sangrados tambiénopcionales en todo caso y nos sirven sólo para ver el código de una manera más ordenada. Mantener el código bien estructurado y escrito de una manera comprensible es muy importante, ya que nos hará la vida más agradable a la hora de programar y más adelante cuando tengamos que revisar los programas. En este manual utilizaré una notación como la que has podido ver en las líneas anteriores, y verás en adelante, además mantendré esa notación en todo momento. Esto sin lugar a dudas hará que los códigos con ejemplos sean comprensibles más rápidamente, si no lo hiciéramos así sería un verdadero incordio leerlos. Esta misma receta es aplicable a los códigos que has de crear tú y el principal beneficiado serás tú mismo y los compañeros que lleguen a leer tu código.

Operador IF

Hay un operador que no hemos visto todavía y es una forma más esquemática de realizar algunos IF sencillos. Proviene del lenguaje C, donde se escriben muy pocas líneas de código que resulta muy elegante. Este operador es un claro ejemplo de ahorro de líneas y caracteres al escribir los scripts. Lo veremos rápidamente, pues la única razón por la que lo incluyo es para que sepas que existe y si lo encuentras en alguna ocasión por ahí sepas identificarlo y cómo funciona.

Un ejemplo de uso del operador IF se puede ver a continuación.

```
Variable = (condición) ? valor1 : valor2
```

Este ejemplo no sólo realiza una comparación de valores, además asigna un valor a una variable. Lo que hace es evaluar la condición (colocada entre paréntesis) y si es positiva asigna el valor1 a la variable y en caso contrario le asigna el valor2. Veamos un ejemplo:

```
momento = (hora_actual < 12) ? "Antes del mediodía" : "Después del mediodía"
```

Este ejemplo mira si la hora actual es mayor que 12. Si es así, es que ahora es antes del mediodía, así que asigna "Antes del mediodía" a la variable momento. Si la hora es mayo o igual a 12 es que ya es después de mediodía, con lo que se asigna el texto "Después del mediodía" a la variable momento.

Estructura SWITCH

Utilizada para tomar decisiones en función de distintos estados de las variables.

Es la otra expresión disponible en Javascript para tomar decisiones en función de distintos estados de las variables. Esta expresión se utiliza cuando tenemos múltiples posibilidades como resultado de la evaluación de una sentencia.

La estructura SWITCH se incorporó a partir de la versión 1.2 de Javascript (Netscape 4 e Internet Explorer 4). Su sintaxis es la siguiente.

```
switch (expresión) {  
    case valor1:  
        Sentencias a ejecutar si la expresión tiene como valor a valor1  
        break  
    case valor2:  
        Sentencias a ejecutar si la expresión tiene como valor a valor2  
        break  
    case valor3:  
        Sentencias a ejecutar si la expresión tiene como valor a valor3  
        break  
    default:  
        Sentencias a ejecutar si el valor no es ninguno de los anteriores  
}
```

La expresión se evalúa, si vale valor1 se ejecutan las sentencias relacionadas con ese caso. Si la expresión vale valor2 se ejecutan las instrucciones relacionadas con ese valor y así sucesivamente, por tantas opciones como deseemos. Finalmente, para todos los casos no contemplados anteriormente se ejecuta el caso por defecto.

La palabra break es opcional, pero si no la ponemos a partir de que se encuentre coincidencia con un valor se ejecutarán todas las sentencias relacionadas con este y todas las siguientes. Es decir, si en nuestro esquema anterior no hubiese ningún break y la expresión valiese valor1, se ejecutarían las sentencias relacionadas con valor1 y también las relacionadas con valor2, valor3 y default.

También es opcional la opción default u opción por defecto.

Veamos un ejemplo de uso de esta estructura. Supongamos que queremos indicar que día de la semana es. Si el día es 1 (lunes) sacar un mensaje indicándolo, si el día es 2 (martes) debemos sacar un mensaje distinto y así sucesivamente para cada día de la semana, menos en el 6 (sábado) y 7 (domingo) que queremos mostrar el mensaje "es fin de semana". Para días mayores que 7 indicaremos que ese día no existe.

```
Switch (dia_de_la_semana) {  
    case 1:
```

```

document.write("Es Lunes")
break
case 2:
    document.write("Es Martes")
    break
case 3:
    document.write("Es Miércoles")
    break
case 4:
    document.write("Es Jueves")
    break
case 5:
    document.write("Es viernes")
    break
case 6:
case 7:
    document.write("Es fin de semana")
    break
default:
    document.write("Ese día no existe")
}

```

El ejemplo es relativamente sencillo, solamente puede tener una pequeña dificultad, consistente en interpretar lo que pasa en el caso 6 y 7, que habíamos dicho que teníamos que mostrar el mismo mensaje. En el caso 6 en realidad no indicamos ninguna instrucción, pero como tampoco colocamos un break se ejecutará la sentencia o sentencias del caso siguiente, que corresponden con la sentencia indicada en el caso 7 que es el mensaje que informa que es fin de semana. Si el caso es 7 simplemente se indica que es fin de semana, tal como se pretendía.

Bucle FOR

Descripción y ejemplos de funcionamiento del bucle FOR.

El bucle FOR se utiliza para repetir más instrucciones un determinado número de veces. De entre todos los bucles, el FOR se **suele utilizar cuando sabemos seguro el número de veces que queremos que se ejecute la sentencia.** La sintaxis del bucle se muestra a continuación.

```

for (initialización; condición; actualización) {
    sentencias a ejecutar en cada iteración
}

```

El bucle FOR tiene tres partes incluidas entre los paréntesis. La primera es la inicialización, que se ejecuta solamente al comenzar la primera iteración del bucle. En esta parte se suele colocar la variable que utilizaremos para llevar la cuenta de las veces que se ejecuta el bucle.

La segunda parte es la condición, que se evaluará cada vez que comience la iteración del bucle. Contiene una expresión para comprobar cuándo se ha de detener el bucle, o mejor dicho, la condición que se debe cumplir para que continúe la ejecución del bucle.

Por último tenemos la actualización, que sirve para indicar los cambios que queramos ejecutar en las variables cada vez que termina la iteración del bucle, antes de comprobar si se debe seguir ejecutando.

Después del for se colocan las sentencias que queremos que se ejecuten en cada iteración, acotadas entre llaves.

Un ejemplo de utilización de este bucle lo podemos ver a continuación, donde se imprimirán los números del 0 al 10.

```
var i  
for (i=0;i<=10;i++) {  
    document.write(i)  
}
```

En este caso se inicializa la variable i a 0. Como condición para realizar una iteración, se tiene que cumplir que la variable i sea menor o igual que 10. Como actualización se incrementará en 1 la variable i.

Como se puede comprobar, **este bucle es muy potente, ya que en una sola línea podemos indicar muchas cosas distintas y muy variadas.**

Por ejemplo si queremos escribir los número del 1 al 1.000 de dos en dos se escribirá el siguiente bucle.

```
for (i=1;i<=1000;i+=2)  
    document.write(i)
```

Si nos fijamos, en cada iteración actualizamos el valor de i incrementándolo en 2 unidades.

Otro detalle, no utilizamos las llaves englobando las instrucciones del bucle FOR porque sólo tiene una sentencia y en este caso no es obligado, tal como pasaba con las instrucciones del IF.

Si queremos contar descendentemente del 343 al 10 utilizaríamos este bucle.

```
for (i=343;i>=10;i--)  
    document.write(i)  
}
```

En este caso decrementamos en una unidad la variable i en cada iteración.

Ejemplo

Vamos a hacer una pausa para asimilar el bucle for con un ejercicio que no encierra ninguna dificultad si hemos entendido el funcionamiento del bucle.

Se trata de hacer un bucle que escriba en una página web los encabezamientos desde <H1> hasta <H6> con un texto que ponga "Encabezado de nivel x".

Lo que deseamos escribir en una página web mediante Javascript es lo siguiente:

```
<H1>Encabezado de nivel 1</H1>
```

```
<H2>Encabezado de nivel 2</H2>
<H3>Encabezado de nivel 3</H3>
<H4>Encabezado de nivel 4</H4>
<H5>Encabezado de nivel 5</H5>
<H6>Encabezado de nivel 6</H6>
```

Para ello tenemos que hacer un bucle que empiece en 1 y termine en 6 y en cada iteración escribiremos el encabezado que toca.

```
for (i=1;i<=6;i++) {
    document.write("<H" + i + ">Encabezado de nivel " + i + "</H" + i + ">")
```

Este script se puede [ver en funcionamiento aquí](#).

Bucles WHILE y DO WHILE

Descripción y diferentes usos de los dos tipos de bucles WHILE con algunos ejemplos.

Veamos ahora los dos tipos de bucles WHILE que podemos utilizar en Javascript y los usos de cada uno.

Bucle WHILE

Estos bucles se utilizan cuando queremos repetir la ejecución de unas sentencias un número indefinido de veces, siempre que se cumpla una condición. Se más sencillo de comprender que el bucle FOR, pues no incorpora en la misma línea la inicialización de las variables su condición para seguir ejecutándose y su actualización. Sólo se indica, como veremos a continuación, la condición que se tiene que cumplir para que se realice una iteración.

```
while (condición){
    sentencias a ejecutar
}
```

Un ejemplo de código donde se utiliza este bucle se puede ver a continuación.

```
var color = ""
while (color != "rojo")
    color = dame un color
}
```

Este es un ejemplo de lo más sencillo que se puede hacer con un bucle while. Lo que hace es pedir que el usuario introduzca un color mientras que el color no sea rojo. Para ejecutar un bucle como este primero tenemos que inicializar la variable que vamos utilizar en la condición de iteración del bucle. Con la variable inicializada podemos escribir el bucle, que comprobará para ejecutarse que el la variable color sea distinto de "rojo". En cada iteración del bucle se pide un nuevo color al usuario para actualizar la variable color y se termina la iteración, con lo que retornamos al principio del bucle, donde tenemos que volver a evaluar si lo que hay en la variable

color es "rojo" y así sucesivamente mientras que no se haya introducido como color el texto "rojo". Obviamente la expresión dame un color no es Javascript, pero como no sabemos todavía cómo escribir eso en Javascript es mejor verlo más adelante.

Bucle DO...WHILE

Es el último de los bucles que hay en Javascript. Se utiliza generalmente cuando no sabemos cuantas veces se habrá de ejecutar el bucle, igual que el bucle WHILE, con la diferencia de que sabemos seguro que el bucle por lo menos se ejecutará una vez.

Este tipo de bucle se introdujo en Javascript 1.2, por lo que no todos los navegadores los soportan, sólo los de versión 4 o superior. En cualquier caso, cualquier código que quieras escribir con DO...WHILE se puede escribir también utilizando un bucle WHILE, con lo que en navegadores antiguos deberás traducir tu bucle DO...WHILE por un bucle WHILE.

La sintaxis es la siguiente.

```
do {  
    sentencias del bucle  
} while (condición)
```

El bucle se ejecuta siempre una vez y al final se evalúa la condición para decir si se ejecuta otra vez el bucle o se termina su ejecución.

Veamos el ejemplo que escribimos para un bucle WHILE en este otro tipo de bucle.

```
var color  
do {  
    color = dame un color  
} while (color != "rojo")
```

Este ejemplo funciona exactamente igual que el anterior, excepto que no tuvimos que inicializar la variable color antes de introducirnos en el bucle. Pide un color mientras que el color introducido es distinto que "rojo".

Ejemplo

Vamos a ver a continuación un ejemplo más práctico sobre cómo trabajar con un bucle WHILE. Como resulta muy difícil hacer ejemplos prácticos con lo poco que sabemos sobre Javascript, vamos a adelantar una instrucción que aun no conocemos.

En este ejemplo vamos a declarar una variable e inicializarla a 0. Luego iremos sumando a esa variable un número aleatorio del 1 al 100 hasta que sumemos 1.000 o más, imprimiendo el valor de la variable suma después de cada operación. Será necesario utilizar el bucle WHILE porque no sabemos exactamente el número de iteraciones que tendremos que realizar.

```
var suma = 0  
while (suma < 1000){
```

```

suma += parseInt(Math.random() * 100)
document.write (suma + "<br>")
}

```

Suponemos que por lo que respecta al bucle WHILE no habrá problemas, pero donde si que puede haberlos es en la sentencia utilizada para tomar un número aleatorio. Sin embargo, no es necesario explicar aquí la sentencia porque lo tenemos planeado hacer más adelante.

Podemos [ver una página con el ejemplo en funcionamiento](#).

Break y continue

Dos instrucciones que aumentan el control sobre los bucles. Sirven para pararlos o continuar con la siguiente iteración.

De manera adicional al uso de las distintas estructuras de bucle se pueden utilizar dos instrucciones para

- Detener la ejecución de un bucle y salirse de él
- Detener la iteración actual y volver al principio del bucle.

Son las instrucciones break y continue.

Break

Se detiene un bucle utilizando la palabra break. Detener un bucle significa salirse de él y dejarlo todo como está para continuar con el flujo del programa inmediatamente después del bucle.

```

for (i=0;i<10;i++){
  document.write (i)
  escribe = dime si continúo
  if (escribe == "no")
    break
}

```

Este ejemplo escribe los números del 0 al 9 y en cada iteración del bucle pregunta al usuario si desea continuar. Si el usuario dice cualquier cosa continua excepto cuando dice "no" que entonces se sale del bucle y deja la cuenta por donde se había quedado.

Continue

Sirve para volver al principio del bucle en cualquier momento, sin ejecutar las líneas que haya por debajo de la palabra continue.

```

var i=0
while (i<7){
  incrementar = dime si incremento
  if (incrementar == "no")

```

```

        continue
    ++
}
```

Este ejemplo, en condiciones normales contaría hasta desde i=0 hasta i=7, pero cada vez que se ejecuta el bucle pregunta al usuario si desea incrementar la variable o no. Si introduce "no" se ejecuta la sentencia continue, con lo que se vuelve al principio del bucle sin llegar a incrementar en 1 la variable i, ya que se ignoran las sentencias que hayan por debajo del continue.

Ejemplo

Un ejemplo más práctico sobre estas instrucciones se puede ver a continuación. Se trata de un bucle FOR planeado para llegar hasta 1.000 pero que lo vamos a parar con break cuando lleguemos a 333.

```

for (i=0;i<=1000;i++){
    document.write(i + "<br>")
    if (i==333)
        break;
}
```

Podemos [ver una página con el ejemplo en funcionamiento](#).

Bucles anidados en Javascript

Explicamos lo que es un bucle anidado, cómo funcionan y para qué sirven. Vemos algunos ejemplos.

Anidar un bucle consiste en meter ese bucle dentro de otro. La anidación de bucles es necesaria para hacer determinados procesamientos un poco más complejos que los que hemos visto en los ejemplos anteriores y seguro que en vuestra experiencia como programadores los habréis utilizado ya o los utilizaréis en un futuro.

Un bucle anidado tiene una estructura como la que sigue. Vamos a tratar de explicarlo a la vista de estas líneas:

```

for (i=0;i<10;i++){
    for (j=0;j<10;j++) {
        document.write(i + "-" + j)
    }
}
```

La ejecución funcionará de la siguiente manera. Para empezar se inicializa el primer bucle, con lo que la variable i valdrá 0 y a continuación se inicializa el segundo bucle, con lo que la variable j valdrá también 0. En cada iteración se imprime el valor de la variable i, un guión ("") y el valor de la variable j, como las dos variables valen 0, se imprimirá el texto "0-0" en la página web.

El bucle que está anidado (más hacia dentro) es el que más veces se ejecuta, en

en este ejemplo, para cada iteración del bucle más externo el bucle anidado se ejecutará por completo una vez, es decir, hará sus 10 iteraciones. En la página web se escribirían estos valores, en la primera iteración del bucle externo y desde el principio:

0-0
0-1
0-2
0-3
0-4
0-5
0-6
0-7
0-8
0-9

Para cada iteración del bucle externo se ejecutarán las 10 iteraciones del bucle interno o anidado. Hemos visto la primera iteración, ahora vamos a ver las siguientes iteraciones del bucle externo. En cada una acumula una unidad en la variable i, con lo que saldrían estos valores.

1-0
1-1
1-2
1-3
1-4
1-5
1-6
1-7
1-8
1-9

Y luego estos.

2-0
2-1
2-2
2-3
2-4
2-5
2-6
2-7
2-8
2-9

Así hasta que se terminen los dos bucles, que sería cuando se alcanzase el valor 9-9.

Veamos un ejemplo muy parecido al anterior, aunque un poco más útil. Se trata de imprimir en la página las todas las tablas de multiplicar. Del 1 al 9, es decir, la tabla del 1, la del 2, del 3...

```
for (i=1;i<10;i++){  
    document.write("<br><b>La tabla del " + i + ":</b><br>")  
    for (j=1;j<10;j++) {
```

```

        document.write(i + " x " + j + ": ")
        document.write(i*j)
        document.write("<br>")
    }
}

```

Con el primer bucle controlamos la tabla actual y con el segundo bucle la desarrollamos. En el primer bucle escribimos una cabecera, en negrita, indicando la tabla que estamos escribiendo, primero la del 1 y luego las demás en orden ascendente hasta el 9. Con el segundo bucle escribo cada uno de los valores de cada tabla. Se puede [ver el ejemplo en marcha en este enlace](#).

Veremos más cosas con bucles anidados en capítulos posteriores, aunque si queremos adelantarnos un poco para ver un nuevo ejemplo que afiance estos conocimientos podemos ir viendo un [ejemplo en el Taller de Javascript sobre bucles anidados](#), donde se construye la tabla con todos los colores puros en definiciones de 256 colores.

Funciones en Javascript

Definimos el concepto de función y aprendemos a crearlas y también llamarlas.

Ahora vamos a ver un tema muy importante, sobretodo para los que no han programado nunca y con Javascript están dando sus primeros pasos en el mundo de la programación ya que veremos un concepto nuevo, el de función, y los usos que tiene. Para los que ya conoczan el concepto de función también será un capítulo útil, pues también veremos la sintaxis y funcionamiento de las funciones en Javascript.

Qué es una función

A la hora de hacer un programa ligeramente grande existen determinados procesos que se pueden concebir de forma independiente, y que son más sencillos de resolver que el problema entero. Además, estos suelen ser realizados repetidas veces a lo largo de la ejecución del programa. Estos procesos se pueden agrupar en una función, definida para que no tengamos que repetir una y otra vez ese código en nuestros scripts, sino que simplemente llamamos a la función y ella se encarga de hacer todo lo que debe.

Así que podemos ver una función como una serie de instrucciones que englobamos dentro de un mismo proceso. Este proceso se podrá luego ejecutar desde cualquier otro sitio con solo llamarlo. Por ejemplo, en una página web puede haber una función para cambiar el color del fondo y desde cualquier punto de la página podríamos llamarla para que nos cambie el color cuando lo deseemos.

Las funciones se utilizan constantemente, no sólo las que escribes tu, sino también las que ya están definidas en el sistema, pues todos los lenguajes de programación tienen un montón de funciones para realizar procesos habituales como por ejemplo obtener la hora, imprimir un mensaje en la pantalla o convertir variables de un tipo a otro. Ya hemos visto alguna función en nuestros sencillos ejemplos anteriores cuando hacíamos un `document.write()` en realidad estábamos llamando a la función

`write()` asociada al documento de la página que escribe un texto en la página. En los capítulos de funciones vamos primero a ver cómo realizar nuestras propias funciones y cómo llamarlas luego. A lo largo del libro veremos muchas de las funciones definidas en Javascript que debemos utilizar para realizar distintos tipos de acciones habituales.

Cómo se escribe una función

Una función se debe definir con una sintaxis especial que vamos a conocer a continuación.

```
function nombrefuncion (){  
    instrucciones de la función  
    ...  
}
```

Primero se escribe la palabra `function`, reservada para este uso. Seguidamente se escribe el nombre de la función, que como los nombres de variables puede tener números, letras y algún carácter adicional como en guión bajo. A continuación se colocan entre llaves las distintas instrucciones de la función. Las llaves en el caso de las funciones no son opcionales, además es útil colocarlas siempre como se ve en el ejemplo, para que se vea fácilmente la estructura de instrucciones que engloba la función.

Veamos un ejemplo de función para escribir en la página un mensaje de bienvenida dentro de etiquetas `<H1>` para que quede más resaltado.

```
function escribirBienvenida(){  
    document.write("<H1>Hola a todos</H1>")  
}
```

Simplemente escribe en la página un texto, es una función tan sencilla que el ejemplo no expresa suficientemente el concepto de función, pero ya veremos otras más complejas. Las etiquetas H1 no se escriben en la página, sino que son interpretadas como el significado de la misma, en este caso que escribimos un encabezado de nivel 1. Como estamos escribiendo en una página web, al poner etiquetas HTML se interpretan como lo que son.

Cómo llamar a una función

Cuando se llaman a las funciones Para ejecutar una función la tenemos que llamar en cualquier parte de la página, con eso conseguiremos que se ejecuten todas las instrucciones que tiene la función entre las dos llaves. Para ejecutar la función utilizamos su nombre seguido de los paréntesis.

Dónde colocamos las funciones

Vemos la manera de insertar las funciones Javascript de cliente dentro de las páginas web.

En principio, podemos colocar las funciones en cualquier parte de la página, siempre entre etiquetas <SCRIPT>, claro está. No obstante existe una limitación a la hora de colocarla con relación a los lugares desde donde se la llame. Lo más normal es colocar la función antes de cualquier llamada a la misma y así seguro que nunca nos equivocaremos.

En concreto, la función se debe definir en el bloque <SCRIPT> donde esté la llamada a la función, aunque es indiferente si la llamada se encuentra antes o después la función, dentro del mismo bloque <SCRIPT>.

```
<SCRIPT>
miFuncion()
function miFuncion(){
    //hago algo...
    document.write("Esto va bien")
}
</SCRIPT>
```

Este ejemplo funciona correctamente porque la función está declarada en el mismo bloque que su llamada.

También es válido que la función se encuentre en un bloque <SCRIPT> anterior al bloque donde está la llamada.

```
<HTML>
<HEAD>
    <TITLE>MI PÁGINA</TITLE>
<SCRIPT>
function miFuncion(){
    //hago algo...
    document.write("Esto va bien")
}
</SCRIPT>
</HEAD>
<BODY>

<SCRIPT>
miFuncion()
</SCRIPT>

</BODY>
</HTML>
```

Vemos un código completo sobre cómo podría ser una página web donde las funciones están en la cabecera. Un lugar muy bueno donde colocarlas, porque se supone que en la cabecera no se van a utilizar todavía y siempre podremos disfrutar de ellas en el cuerpo porque ya han sido declaradas seguro.

Esto último en cambio sería un error.

Lo que será un error es una llamada a una función que se encuentra declarada en un bloque <SCRIPT> posterior.

```
<SCRIPT>
miFuncion()
</SCRIPT>

<SCRIPT>
function miFuncion(){
    //hago algo...
    document.write("Esto va bien")
}
</SCRIPT>
```

Parámetros de las funciones

Vemos lo que son los parámetros en llamadas a funciones y cómo usarlos.

Las estructuras que hemos visto anteriormente sobre funciones no son las únicas que debemos aprender para manejarlas en toda su potencia. Las funciones también tienen una entrada y una salida, que se pueden utilizar para recibir y devolver datos.

Parámetros

Los parámetros se usan para mandar valores a la función, con los que ella trabajará para realizar las acciones. Son los valores de entrada que recibe una función. Por ejemplo, una función que realizase una suma de dos números tendría como parámetros a esos dos números. Los dos números son la entrada, así como la salida sería el resultado, pero eso lo veremos más tarde.

Veamos un ejemplo anterior en el que creábamos una función para mostrar un mensaje de bienvenida en la página web, pero al que ahora le vamos a pasar un parámetro que contendrá el nombre de la persona a la que hay que saludar.

```
function escribirBienvenida(nombre){
    document.write("<H1>Hola " + nombre + "</H1>")
}
```

Como podemos ver en el ejemplo, para definir en la función un parámetro tenemos que poner el nombre de la variable que va a almacenar el dato que le pasemos. Esta variable, que en este caso se llama nombre, tendrá como valor el dato que le pasemos a la función cuando la llamemos, además, la variable tendrá vida durante la ejecución de la función y dejará de existir cuando la función termine su ejecución.

Para llamar a una función que tiene parámetros se coloca entre paréntesis el valor del parámetro. Para llamar a la función del ejemplo habría que escribir:

```
escribirBienvenida("Alberto García")
```

Al llamar a la función así, el parámetro nombre toma como valor "Alberto García" y al escribir el saludo por pantalla escribirá "Hola Alberto García" entre etiquetas <H1>.

Los parámetros pueden recibir cualquier tipo de datos, numérico, textual, booleano o un objeto. Realmente no especificamos el tipo del parámetro, por eso debemos tener un cuidado especial al definir las acciones que realizamos dentro de la función y al pasarle valores a la función para asegurarnos que todo es consecuente con los tipos de nuestras variables o parámetros.

Múltiples parámetros

Una función puede recibir tantos parámetros como queramos y para expresarlo se colocan los parámetros separados por comas dentro de los paréntesis. Veamos rápidamente la sintaxis para que la función de antes reciba dos parámetros, el primero el nombre al que saludar y el segundo el color del texto.

```
function escribirBienvenida(nombre,colorTexto){  
    document.write("<FONT color=" + colorTexto + ">")  
    document.write("<H1>Hola " + nombre + "</H1>")  
    document.write("</FONT>")  
}
```

Llamaríamos a la función con esta sintaxis. Entre los paréntesis colocaremos los valores de los parámetros.

```
var miNombre = "Pepe"  
var miColor = "red"  
escribirBienvenida(miNombre,miColor)
```

He colocado entre los paréntesis dos variables en lugar de dos textos entrecomillados. Cuando colocamos variables entre los parámetros en realidad lo que estamos pasando a la función son los valores que contienen las variables y no las mismas variables.

Parámetros se pasan por valor

Al hilo del uso de parámetros en nuestros programas Javascript tenemos que indicar que los parámetros de las funciones se pasan por valor. Esto quiere decir que aunque modifiquemos un parámetro en una función la variable original que habíamos pasado no cambiará su valor. Se puede ver fácilmente con un ejemplo.

```
function pasoPorValor(miParametro){  
    miParametro = 32  
    document.write("he cambiado el valor a 32")  
}  
var miVariable = 5  
pasoPorValor(miVariable)  
document.write ("el valor de la variable es: " + miVariable)
```

En el ejemplo tenemos una función que recibe un parámetro y que modifica el valor del parámetro asignándole el valor 32. También tenemos una variable, que inicializamos a 5 y posteriormente llamamos a la función pasándole esta variable

como parámetro. Como dentro de la función modificamos el valor del parámetro podría pasar que la variable original cambiase de valor, pero como los parámetros no modifican el valor original de las variables esta no cambia de valor. De este modo, al imprimir en pantalla el valor de miVariable se imprimirá el número 5, que es el valor original de la variable, en lugar de 32 que era el valor col el que habíamos actualizado el parámetro.

En javascript sólo se pueden pasar las variables por valor.

Valores de retorno

Las funciones pueden devolver valores, vemos cómo. También vemos un apunte sobre el ámbito de variables en funciones en Javascript.

Las funciones también pueden retornar valores, de modo que al ejecutar la función se podrá realizar acciones y dar un valor como salida. Por ejemplo, una función que calcula el cuadrado de un número tendrá como entrada -tal como vimos- a ese número y como salida tendrá el valor resultante de hallar el cuadrado de ese número. Una función que devuelva el día de la semana tendría como salida en día de la semana.

Veamos un ejemplo de función que calcula la media de dos números. La función recibirá los dos números y retornará el valor de la media.

```
function media(valor1,valor2){  
    var resultado  
    resultado = (valor1 + valor2) / 2  
    return resultado  
}
```

Para especificar el valor que retornará la función se utiliza la palabra return seguida del valor que se desea devolver. En este caso se devuelve el contenido de la variable resultado, que contiene la media de los dos números.

Para recibir los valores que devuelve una función se coloca el operador de asignación =. Para ilustrar esto se puede ver este ejemplo, que llamará a la función media() y guardará el resultado de la media en una variable para luego imprimirla en la página.

```
var miMedia  
miMedia = media(12,8)  
document.write (miMedia)
```

Múltiples return

En una misma función podemos colocar más de un return. Lógicamente sólo vamos a poder retornar una cosa, pero dependiendo de lo que haya sucedido en la función podrá ser de un tipo u otro, con unos datos u otros.

En esta función podemos ver un ejemplo de utilización de múltiples return. Se trata de una función que devuelve un 0 si el parámetro recibido era par y el valor del parámetro si este era impar.

```

function multipleReturn(numero){
    var resto = numero % 2
    if (resto == 0)
        return 0
    else
        return numero
}

```

Para averiguar si un número es par hallamos el resto de la división al dividirlo entre 2. Si el resto es cero es que era par y devolvemos un 0, en caso contrario -el número es impar- devolvemos el parámetro recibido.

Ámbito de las variables en funciones

Dentro de las funciones podemos declarar variables, incluso los parámetros son como variables que se declaran en la cabecera de la función y que se inicializan al llamar a la función. Todas las variables declaradas en una función son locales a esa función, es decir, solo tendrán validez durante la ejecución de la función.

Podemos declarar variables en funciones que tengan el mismo nombre que una variable global a la página. Entonces, dentro de la función la variable que tendrá validez es la variable local y fuera de la función tendrá validez la variable global a la página.

En cambio, si no declaramos las variables en las funciones se entenderá por javascript que estamos haciendo referencia a una variable global a la página, de modo que si no está creada la variable la crea, pero siempre global a la página en lugar de local a la función.

Arrays en Javascript

Vemos que son los arrays, para qué sirven y cómo utilizarlos.

En los lenguajes de programación existen estructuras de datos especiales que nos sirven para guardar información más compleja que simples variables. Una estructura típica en todos los lenguajes es el Array, que es como una variable donde podemos introducir varios valores, en lugar de solamente uno como ocurre con las variables normales.

Los arrays nos permiten guardar varias variables y acceder a ellas de manera independiente, es como tener una variable con distintos comportamientos donde podemos introducir datos distintos. Para ello utilizamos un índice que nos permite especificar el comportamiento o posición a la que nos estamos refiriendo.

Los arrays se introdujeron en versiones Javascript 1.1 o superiores, es decir, solo los podemos utilizar a partir de los navegadores 3.0. Para navegadores antiguos se puede simular el array utilizando sintaxis de programación orientada a objetos, pero dada la complejidad de esta tarea, por lo menos en el momento en que nos encontramos y las pocas ocasiones que los deberemos utilizar vamos a ver cómo utilizar el auténtico array de Javascript.

Creación de Arrays

El primer paso para utilizar un array es crearlo. Para ello utilizamos un objeto Javascript ya implementado en el navegador. Veremos en adelante un tema para explicar lo que es la orientación a objetos, aunque no será necesario para poder entender el uso de los arrays. Esta es la sentencia para crear un objeto array:

```
var miArray = new Array()
```

Esto crea un array en la página que esta ejecutándose. El array se crea sin ningún contenido, es decir, no tendrá ninguna casilla o compartimiento creado. También podemos crear el array especificando el número de compartimentos que va a tener.

```
var miArray = new Array(10)
```

En este caso indicamos que el array va a tener 10 posiciones, es decir, 10 casillas donde guardar datos.

Es importante que nos fijemos que la palabra `Array` en código Javascript se escribe con la primera letra en mayúscula. Como en Javascript las mayúsculas y minúsculas si que importan, si lo escribimos en minúscula no funcionará.

Tanto se indique o no el número de casillas del array, podemos introducir en el array cualquier dato. Si la casilla está creada se introduce simplemente y si la casilla no estaba creada se crea y luego se introduce el dato, con lo que el resultado final es el mismo. Esta creación de casillas es dinámica y se produce al mismo tiempo que los scripts se ejecutan. Veamos a continuación cómo introducir valores en nuestros arrays.

```
miArray[0] = 290  
miArray[1] = 97  
miArray[2] = 127
```

Se introducen indicando entre corchetes el índice de la posición donde queríamos guardar el dato. En este caso introducimos 290 en la posición 0, 97 en la posición 1 y 127 en la 2. Los arrays empiezan siempre en la posición 0, así que un array que tenga por ejemplo 10 posiciones, tendrá casillas de la 0 a la 9. Para recoger datos de un array lo hacemos igual: poniendo entre corchetes el índice de la posición a la que queremos acceder. Veamos cómo se imprimiría en la pantalla el contenido de un array.

```
var miArray = new Array(3)  
  
miArray[0] = 155  
miArray[1] = 4  
miArray[2] = 499  
  
for (i=0;i<3;i++){  
    document.write("Posición " + i + " del array: " + miArray[i])  
    document.write("<br>")  
}
```

Hemos creado un array con tres posiciones, luego hemos introducido un valor en cada una de las posiciones del array y finalmente las hemos impreso. En general, el

recorrido por arrays para imprimir sus posiciones o cualquier otra cosa se hace utilizando bucles. En este caso utilizamos un bucle FOR que va desde el 0 hasta el 2.

Podemos [ver el ejemplo en marcha en otra página](#).

Tipos de datos en los arrays

En las casillas de los arrays podemos guardar datos de cualquier tipo. Podemos ver un array donde introducimos datos de tipo carácter.

```
miArray[0] = "Hola"  
miArray[1] = "a"  
miArray[2] = "todos"
```

Incluso, en Javascript podemos guardar distintos tipos de datos en las casillas de un mismo array. Es decir, podemos introducir números en unas casillas, textos en otras, booleanos o cualquier otra cosa que deseemos.

```
miArray[0] = "desarrolloweb.com"  
miArray[1] = 1275  
miArray[1] = 0.78  
miArray[2] = true
```

Longitud de los arrays

Aprendemos más cosas sobre el funcionamiento de los arrays y su propiedad length.

Todos los arrays en javascript, aparte de almacenar el valor de cada una de sus posiciones también almacenan el número de posiciones que tienen. Para ello utilizan una propiedad del objeto array, la propiedad length. Ya veremos en objetos qué es una propiedad, pero para nuestro caso podemos imaginarnos que es como una variable, adicional a las posiciones, que almacena un número igual al número de casillas del array.

Para acceder a una propiedad de un objeto se ha de utilizar el operador punto. Se escribe el nombre del array que queremos acceder al número de posiciones que tiene, sin corchetes ni paréntesis, seguido de un punto y la palabra length.

```
var miArray = new Array()  
  
miArray[0] = 155  
miArray[1] = 499  
miArray[2] = 65  
  
document.write("Longitud del array: " + miArray.length)
```

Este código imprimiría en pantalla el número de posiciones del array, que en este caso es 3. Recordamos que un array con 3 posiciones abarca desde la posición 0 a la 2.

Es muy habitual que se utilice la propiedad length para poder recorrer un array por todas sus posiciones. Para ilustrarlo vamos a ver un ejemplo de recorrido por este array para mostrar sus valores.

```
for (i=0;i<miArray.length;i++){
    document.write(miArray[i])
}
```

Hay que fijarse que el bucle for se ejecuta siempre que i valga menos que la longitud del array, extraída de su propiedad length.

El siguiente ejemplo nos servirá para conocer mejor los recorridos por los arrays, el funcionamiento de la propiedad length y la creación dinámica de nuevas posiciones. Vamos a crear un array con 2 posiciones y rellenar su valor. Posteriormente introduciremos un valor en la posición 5 del array. Finalmente imprimiremos todas las posiciones del array para ver lo que pasa.

```
var miArray = new Array(2)

miArray[0] = "Colombia"
miArray[1] = "Estados Unidos"

miArray[5] = "Brasil"

for (i=0;i<miArray.length;i++){
    document.write("Posición " + i + " del array: " + miArray[i])
    document.write("<br>")
}
```

El ejemplo es sencillo. Se puede apreciar que hacemos un recorrido por el array desde 0 hasta el número de posiciones del array (indicado por la propiedad length). En el recorrido vamos imprimiendo el número de la posición seguido del contenido del array en esa posición. Pero podemos tener una duda al preguntarnos cuál será el número de elementos de este array, ya que lo habíamos declarado con 2 y luego le hemos introducido un tercero en la posición 5. Al ver la salida del programa podremos contestar nuestras preguntas. Será algo parecido a esto:

```
Posición 0 del array: Colombia
Posición 1 del array: Estados Unidos
Posición 2 del array: null
Posición 3 del array: null
Posición 4 del array: null
Posición 5 del array: Brasil
```

Se puede ver claramente que el número de posiciones es 6, de la 0 a la 5. Lo que ha ocurrido es que al introducir un dato en la posición 5, todas las casillas que no estaban creadas hasta la quinta se crean también.

Las posiciones de la 2 a la 4 están sin inicializar. En este caso nuestro navegador ha escrito la palabra null para expresar esto, pero otros navegadores podrán utilizar la palabra undefined. Ya veremos más adelante qué es este null y dónde lo podemos utilizar, lo importante ahora es que comprendas cómo trabajan los arrays y los utilices correctamente.

Podemos [ver el efecto de este script en tu navegador en una página a parte](#).

Arrays multidimensionales

Vemos qué son los arrays multidimensionales y cómo utilizarlos. Además explicamos cómo inicializar arrays en su declaración.

Los arrays multidimensionales son unas estructuras de datos que almacenan los valores en más de una dimensión. Los arrays que hemos visto hasta ahora almacenan valores en una dimensión, por eso para acceder a las posiciones utilizamos tan solo un índice. Los arrays de 2 dimensiones guardan sus valores, por decirlo de alguna manera, en filas y columnas y por ello necesitaremos dos índices para acceder a cada una de sus posiciones.

Dicho de otro modo, un array multidimensional es como un contenedor que guardara más valores para cada posición, es decir, como si los elementos del array fueran a su vez otros arrays.

En Javascript no existe un auténtico objeto array-multidimensional. Para utilizar estas estructuras podremos definir arrays que donde en cada una de sus posiciones habrá otro array. En nuestros programas podremos utilizar arrays de cualquier dimensión, veremos a continuación cómo trabajar con arrays de dos dimensiones, que serán los más comunes.

En este ejemplo vamos a crear un array de dos dimensiones donde tendremos por un lado ciudades y por el otro la temperatura media que hace en cada una durante de los meses de invierno.

```
var temperaturas_medias_ciudad0 = new Array(3)
temperaturas_medias_ciudad0[0] = 12
temperaturas_medias_ciudad0[1] = 10
temperaturas_medias_ciudad0[2] = 11
```

```
var temperaturas_medias_ciudad1 = new Array (3)
temperaturas_medias_ciudad1[0] = 5
temperaturas_medias_ciudad1[1] = 0
temperaturas_medias_ciudad1[2] = 2
```

```
var temperaturas_medias_ciudad2 = new Array (3)
temperaturas_medias_ciudad2[0] = 10
temperaturas_medias_ciudad2[1] = 8
temperaturas_medias_ciudad2[2] = 10
```

Con las anteriores líneas hemos creado tres arrays de 1 dimensión y tres elementos, como los que ya conocíamos. Ahora crearemos un nuevo array de tres elementos e introduciremos dentro de cada una de sus casillas los arrays creados anteriormente, con lo que tendremos un array de arrays, es decir, un array de 2 dimensiones.

```
var temperaturas_cuidades = new Array (3)
temperaturas_cuidades[0] = temperaturas_medias_ciudad0
temperaturas_cuidades[1] = temperaturas_medias_ciudad1
temperaturas_cuidades[2] = temperaturas_medias_ciudad2
```

Vemos que para introducir el array entero hacemos referencia al mismo sin paréntesis ni corchetes, sino sólo con su nombre. El array temperaturas_cuidades es nuestro array bidimensional.

También es interesante ver cómo se realiza un recorrido por un array de dos dimensiones. Para ello tenemos que hacer un recorrido por cada una de las casillas del array bidimensional y dentro de estas hacer un nuevo recorrido para cada una de sus casillas internas. Es decir, un recorrido por un array dentro de otro.

El método para hacer un recorrido dentro de otro es colocar un bucle dentro de otro, lo que se llama un bucle anidado. Puede resultar complicado el hacer un bucle anidado, pero nosotros ya hemos tenido ocasión de [practicar en un capítulo anterior](#). Así que en este ejemplo vamos a meter un bucle FOR dentro de otro. Además, vamos a escribir los resultados en una tabla, lo que complicará un poco el script, pero podremos ver cómo construir una tabla desde javascript a medida que realizamos el recorrido anidado al bucle.

```
document.write("<table width=200 border=1 cellpadding=1 cellspacing=1>");  
for (i=0;i<temperaturas_cuidades.length;i++){  
    document.write("<tr>")  
    document.write("<td><b>Ciudad " + i + "</b></td>")  
    for (j=0;j<temperaturas_cuidades[i].length;j++){  
        document.write("<td>" + temperaturas_cuidades[i][j] + "</td>")  
    }  
    document.write("</tr>")  
}  
document.write("</table>")
```

Este script resulta un poco más complejo que los vistos anteriormente. La primera acción consiste en escribir la cabecera de la tabla, es decir, la etiqueta <TABLE> junto con sus atributos. Con el primer bucle realizamos un recorrido a la primera dimensión del array y utilizamos la variable i para llevar la cuenta de la posición actual. Por cada iteración de este bucle escribimos una fila y para empezar la fila abrimos la etiqueta <TR>. Además, escribimos en una casilla el número de la ciudad que estamos recorriendo en ese momento. Posteriormente ponemos otro bucle que va recorriendo cada una de las casillas del array en su segunda dimensión y escribimos la temperatura de la ciudad actual en cada uno de los meses, dentro de su etiqueta <TD>. Una vez que acaba el segundo bucle se han impreso las tres temperaturas y por lo tanto la fila está terminada. El primer bucle continúa repitiéndose hasta que todas las ciudades están impresas y una vez terminado cerramos la tabla.

Podemos [ver el ejemplo en marcha](#) y examinar el código del script entero.

Inicialización de arrays

Para terminar con el tema de los arrays vamos a ver una manera de inicializar sus valores a la vez que lo declaramos, así podemos realizar de una manera más rápida el proceso de introducir valores en cada una de las posiciones del array.

El método normal de crear un array vimos que era a través del objeto Array, poniendo entre paréntesis el número de casillas del array o no poniendo nada, de modo que el array se crea sin ninguna posición. Para introducir valores a un array se hace igual, pero poniendo entre los paréntesis los valores con los que deseamos rellenar las casillas separados por coma. Veámoslo con un ejemplo que crea un array con los nombres de los días de la semana.

```
var diasSemana = new  
Array("Lunes","Martes","Miércoles","","Jueves","Viernes","Sábado","Domingo")
```

El array se crea con 7 casillas, de la 0 a la 6 y en cada casilla se escribe el dia de la semana correspondiente (Entre comillas porque es un texto).

Ahora vamos a ver algo más complicado, se trata de declarar el array bidimensional que utilizamos antes para las temperaturas de las ciudades en los meses en una sola línea, introduciendo los valores a la vez.

```
var temperaturas_cuidades = new Array(new Array (12,10,11), new  
Array(5,0,2),new Array(10,8,10))
```

En el ejemplo introducimos en cada casilla del array otro array que tiene como valores las temperaturas de una ciudad en cada mes.

Pausa y consejos Javascript

Hacemos una pausa en el manual de Javascript para ofrecer una serie de consejos útiles.

Hasta aquí hemos visto la mayor parte de la sintaxis y forma de funcionar del lenguaje Javascript. Ahora podemos escribir scripts simples que hagan uso de variables, funciones, arrays, estructuras de control y toda clase de operadores. Con todo esto conocemos el lenguaje, pero aun queda mucho camino por delante para dominar Javascript y saber hacer todos los efectos posibles en páginas web, que en definitiva es lo que interesa.

De todos modos, este manual nos lo hemos tomado con mucha calma, con intención de que las personas que no estén familiarizadas con el mundo de la programación puedan también tener acceso al lenguaje y aprendan las técnicas básicas que permitirán afrontar futuros retos en la programación. Esperamos entonces que la marcha del manual haya sido provechosa para los más inexpertos y que ahora puedan entender con facilidad las siguientes lecciones o ejemplos, ya que conocen las bases sobre las que están implementados.

Antes de acabar, vamos a dar una serie de consejos a seguir a la hora de programar nuestros scripts que nos pueden ayudar a hacernos la vida más fácil. Algunos consejos son nuevos e importantes, otros se han señalado con anterioridad, pero sin duda viene bien recordar.

Distintos navegadores

Lo primero importante en señalar es que Javascript es un lenguaje muy dinámico y que ha sido implementado poco a poco, a medida que salían nuevos navegadores. Si pensamos en el Netscape 2, el primer navegador que incluía Javascript, y el Netscape 6 o Internet Explorer 6 nos daremos cuenta que han pasado un mundo de novedades entre ellos. Javascript ha cambiado por lo menos tanto como los navegadores y eso representa un problema para los programadores, porque tienen que estar al tanto de las distintas versiones y la manera de funcionar que tienen.

Las bases de javascript, sobre las que hemos hablado hasta ahora, no han cambiado prácticamente nada. Sólo en algunas ocasiones, que hemos señalado según las conocíamos -como los arrays por ejemplo-, el lenguaje ha evolucionado un poco. Sin embargo, a medida que nuevas tecnologías como el DHTML se desarrollaban, los navegadores han variado su manera de manejarlas.

Nuestro consejo es que estéis al tanto de las cosas que funcionan en unos u otros sistemas y que programéis para que vuestras páginas sean compatibles en el mayor número de navegadores. Para procurar esto último es muy aconsejable probar las páginas en varias plataformas distintas. También es muy útil dejar de lado los últimos avances, es decir, no ir a la última, sino ser un poco conservadores, para que las personas que han actualizado menos el browser puedan también visualizar los contenidos.

Consejos para hacer un código sencillo y fácil de mantener

Ahora vamos a dar una serie de consejos para que el código de nuestros scripts sea más claro y libre de errores. Muchos de ellos ya los hemos señalado y somos libres de aplicarlos o no, pero si lo hacemos nuestra tarea como programadores puede ser mucho más agradable, no sólo hoy, sino también el día que tengamos que revisar los scripts en su mantenimiento.

- Utiliza comentarios habitualmente para que lo que estás haciendo en los scripts pueda ser recordado por ti y cualquier persona que tenga que leerlos más adelante.
- Ten cuidado con el ámbito de las variables, recuerda que existen variables globales y locales, que incluso pueden tener los mismos nombres y conoce en cada momento la variable que tiene validez.
- Escribe un código con suficiente claridad, que se consigue con saltos de línea después de cada instrucción, un sangrado adecuado (poner márgenes a cada línea para indicar en qué bloque están incluidas), utilizar las llaves {} siempre, aunque no sean obligatorias y en general mantener siempre el mismo estilo a la hora de programar.
- Aplica un poco de consistencia al manejo de variables. Declara las variables con var. No cambies el tipo del dato que contiene (si era numérica no pongas luego texto, por ejemplo). Dales nombres comprensibles para saber en todo momento qué contienen. Incluso a la hora de darles nombre puedes aplicar una norma, que se trata de que indiquen en sus nombres el tipo de dato que contienen. Por ejemplo, las variables de texto pueden empezar por una s (de String), las variables numéricas pueden empezar por una n o las booleanas por una b.
- Prueba tus scripts cada poco a medida que los vas programando. Puedes escribir un trozo de código y probarlo antes de continuar para ver que todo funciona correctamente. Es más fácil encontrar los errores de código en bloques pequeños que en bloques grandes.

Tratamiento de errores en javascript

Vamos a explicar los errores comunes que podemos cometer y cómo evitarlos y depurarlos. Además veremos una pequeña conclusión a la primera parte del manual.

Para acabar la primera parte del manual de javascript vamos a explicar los errores comunes que podemos cometer y cómo evitarlos y depurarlos. Además veremos una pequeña conclusión a esta parte del manual.

Errores comunes

Cuando ejecutamos los scripts pueden ocurrir dos tipos de errores de sintaxis o de ejecución, los vemos a continuación.

Errores de sintaxis ocurren por escribir de manera errónea las líneas de código, equivocarse a la hora de escribir el nombre de una estructura, utilizar incorrectamente las llaves o los paréntesis o cualquier cosa similar. Estos errores los indica javascript a medida que está cargando los scripts en memoria, lo que hace siempre antes de ejecutarlos, como se indicó en los primeros capítulos. Cuando el analizador sintáctico de javascript detecta un error de estos se presenta el mensaje de error.

Errores de ejecución ocurren cuando se están ejecutando los scripts. Por ejemplo pueden ocurrir cuando llamamos a una función que no ha sido definida. javascript no indica estos errores hasta que no se realiza la llamada a la función.

La manera que tiene javascript de mostrar un error puede variar de un navegador a otro. En versiones antiguas se mostraba una ventanita con el error y un botón de aceptar, tanto en Internet Explorer como en Netscape. En la actualidad los errores de javascript permanecen un poco más ocultos al usuario. Esto viene bien, porque si nuestras páginas tienen algún error en alguna plataforma no será muy molesto para el usuario que en muchas ocasiones ni se dará cuenta. Sin embargo para el programador puede ser un poco más molesto de revisar y se necesitará conocer la manera de que se muestren los errores para que puedan ser reparados.

En versiones de Internet Explorer mayores que la 4 se muestra el error en la barra de estado del navegador y se puede ver una descripción más grande del error si le damos un doble click al ícono de alerta amarillo que aparece en la barra de estado. En Netscape aparece también un mensaje en la barra de estado que además nos indica que para mostrar más información debemos teclear "javascript:" en la barra de direcciones (donde escribimos las URL para acceder a las páginas). Con eso conseguimos que aparezca la Consola javascript, que nos muestra todos los errores que se encuentran en las páginas.

También podemos cometer fallos en la programación que hagan que los scripts no funcionen tal y como deseábamos y que javascript no detecta como errores y por lo tanto no muestra ningún mensaje de error.

Por dejarlo claro vamos a ver un ejemplo en el que nuestro programa puede no funcionar como deseamos sin que javascript ofrezca ningún mensaje de error. En este ejemplo trataríamos de sumar dos cifras pero si una de las variables es de tipo texto podríamos encontrarnos con un error.

```
var numero1 = 23  
var numero2 = "42"  
var suma = numero1 + numero2
```

¿Cuánto vale la variable suma? Como muchos ya sabéis, la variable suma vale "2342" porque al intentar sumar una variable numérica y otra textual, se tratan a las dos como si fueran de tipo texto y por lo tanto, el operador + se aplica como una concatenación de cadenas de caracteres. Si no estamos al tanto de esta cuestión podríamos tener un error en nuestro script ya que el resultado no es el esperado y además el tipo de la variable suma no es numérico sino cadena de

caracteres.

Evitar errores comunes

Vamos a ver ahora una lista de los errores típicos cometidos por inexpertos en la programación en general y en javascript en particular, y por no tan inexpertos.

Utilizar = en expresiones condicionales en lugar de == es un error difícil de detectar cuando se comete, si utilizamos un solo igual lo que estamos haciendo es una asignación y no una comparación para ver si dos valores son iguales.

No conocerse la precedencia de operadores y no utilizar paréntesis para agrupar las operaciones que se desea realizar. En este caso nuestras operaciones pueden dar resultados no deseados.

Usar comillas dobles y simples erróneamente. Recuerda que se pueden utilizar comillas dobles o simples indistintamente, con la norma siguiente: dentro de comillas dobles se deben utilizar comillas simples y viceversa.

Olvidarse de cerrar una llave o cerrar una llave de más.

Colocar varias sentencias en la misma línea sin separarlas de punto y coma. Esto suele ocurrir en los manejadores de eventos, como onclick, que veremos más adelante.

Utilizar una variable antes de inicializarla o no declarar las variables con var antes de utilizarlas también son errores comunes. Recuerda que si no la declaras puedes estar haciendo referencia a una variable global en lugar de una local.

Contar las posiciones de los arrays a partir de 1. Recuerda que los arrays empiezan por la posición 0.

Depurar errores javascript

Cualquier programa es susceptible de contener errores. javascript nos informará de muchos de los errores de la página: los que tienen relación con la sintaxis y los que tienen lugar en el momento de la ejecución de los scripts a causa de equivocarnos al escribir el nombre de una función o una variable. Sin embargo, no son los únicos errores que nos podemos encontrar, también están los errores que ocurren sin que javascript muestre el correspondiente mensaje de error, como vimos anteriormente, pero que hacen que los programas no funcionen como esperábamos.

Para todo tipo de errores, unos más fáciles de detectar que otros, debemos utilizar alguna técnica de depuración que nos ayude a encontrarlos. Lenguajes de programación más potentes que el que tratamos ahora incluyen importantes herramientas de depuración, pero en javascript debemos contentarnos con una rudimentaria técnica. Se trata de utilizar una función predefinida, la función alert() que recibe entre paréntesis un texto y lo muestra en una pequeña ventana que tiene un botón de aceptar.

Con la función alert() podemos mostrar en cualquier momento el contenido de las variables que estamos utilizando en nuestros scripts. Para ello ponemos entre paréntesis la variable que deseamos ver su contenido. Cuando se muestra el contenido de la variable el navegador espera a que apretemos el botón de aceptar y cuando lo hacemos continúa con las siguientes instrucciones del script.

Este es un sencillo ejemplo sobre cómo se puede utilizar la función alert() para mostrar el contenido de las variables.

```
var n_actual = 0
var suma = 0
while (suma<300){
    n_actual ++
    suma += suma + n_actual
    alert("n_actual vale " + n_actual + " y suma vale " + suma)
}
```

Con la función alert() se muestra el contenido de las dos variables que utilizamos en el script. Algo similar a esto es lo que tendremos que hacer para mostrar el contenido de las variables y saber cómo están funcionando nuestros scripts, si todo va bien o hay algún error.

Conclusión

Hasta aquí hemos conocido la sintaxis javascript en profundidad. Aunque aun nos quedan cosas importantes de sintaxis, la visión que has podido tener del lenguaje será suficiente para enfrentarte a los problemas más fundamentales. En adelante presentaremos otros reportajes para aprender a utilizar los recursos con los que contamos a la hora de hacer efectos en páginas web.

Veremos la jerarquía de objetos del navegador, cómo construir nuestros propios objetos, las funciones predefinidas de javascript, características del HTML Dinámico, trabajo con formularios y otras cosas importantes para dominar todas las posibilidades del lenguaje.

Todo ello en nuestro [manual de Javascript II](#) y en el [Taller de Javascript](#).

Introducción al manual II de Javascript

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 faremos 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.

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

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.

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

Vamos a ver algún ejemplo con las funciones más importantes de esta lista.

Ejemplos de funciones de la librería Javascript

Ahora podemos ver varios ejemplos de utilización de funciones de la librería que proporciona 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

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.

Función isNaN

Esta función devuelve un booleano 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 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](#)

Objetos en 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 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 de programación orientada a objetos es imprescindible que nos leamos un [pequeño artículo publicado en DesarrolloWeb sobre este tema](#). Después de su lectura puedes continuar con estas líneas y si conoces ya la programación orientada a objetos continúa leyendo sin pausa.

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.

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()
```

Objetos incorporados en Javascript

Sabiendo ya lo que es la programación orientada a objetos 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 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 booleanos.

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 carácter 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 compuesta 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.

Clase String en Javascript

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.

Propiedades de String

Length

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

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 qué posición se desea que empiece la búsqueda.

lastIndexOf(carácter,desde)

Busca la posición de un carácter exactamente 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 utilizarlo 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 .

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 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.

Ejemplos de funcionamiento de la clase String

Ahora vamos a ver unos ejemplos sobre cómo se utilizan los métodos y propiedades del objeto String.

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](#).

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](#).

Clase Date en Javascript

Sobre este objeto recae todo el trabajo con fechas en Javascript, como obtener una fecha, el día la hora 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 embargo, 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.

Ejemplo de funcionamiento de Date

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 transcurridos 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](#)

Clase Math en Javascript

La clase Math 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 raíz 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.

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.

Métodos de Math

Así mismo, tenemos una serie de métodos para realizar operaciones matemáticas 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 raíz 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 propiedades 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 ("<br>")
```

```
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](#).

Clase Number en Javascript

La clase Number modeliza el tipo de datos numérico. Fue añadido en la versión 1.1 de Javascript (con Netscape Navigator 3). 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 todos los scripts que sirven para hacer las cosas más dispares y útiles.

Nota: conocemos 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](#).

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](#).

Clase Boolean en Javascript

Esta clase nos sirve para crear valores booleanos. Fue añadido en la versión 1.1 de Javascript (con Netscape Navigator 3). Una de sus posibles utilidades es la de conseguir valores booleanos a partir de datos de cualquier otro tipo.

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 booleanos.

Dependiendo de lo que reciba el constructor de la clase Boolean el valor del objeto booleano 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](#).

Creación de clases en 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.

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.

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.

Creación de clases en Javascript II

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 matricula(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.

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.

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)
```

Creación de clases en Javascript III

Para ilustrar el trabajo con objetos y terminar con el ejemplo del AlumnoUniversitario, 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 matricularse 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 matrícula 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 hacemos algún ejemplo adicional.

Jerarquía de objetos del navegador

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.

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")
```

Trabajando con la Jerarquía en Javascript

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.

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

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 jerarquía 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.

Objeto window de Javascript

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.

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

Número 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](#).

Métodos de window en Javascript

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.

Ejemplos de métodos de window

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.

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](#).

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](#).

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 lleno 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.

Objeto document en Javascript

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.

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 . 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 título 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.

Ejemplos de propiedades de document

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

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](#).

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 el 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 días que longanizas"  
onclick="cambiaTitulo('Más días que longanizas')">  
</form>
```

Podemos [ver en funcionamiento este script](#) en otra página web.

Métodos de 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()

Invoca 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>
<br>

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

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

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

Flujo de escritura del documento

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.

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](#).

Trabajo con formularios en Javascript

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 formulario 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.

Ej. trabajo con formularios. 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 posibilidades que tenemos.

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 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.

Propiedades y métodos del objeto form

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

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 submite se actualizará la ventana o frame indicado. Corresponde con el atributo target del formulario.

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"
```

Métodos del objeto form

Estos son los métodos que podemos invocar con un formulario.

submit()

Para hacer que el formulario se submite, 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)

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="validaSubmit()">
</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 validaSubmit(){
    if (document.miFormulario.campo1.value == "") {
        alert("Debe llenar 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 llenar el formulario. En caso de haya algo en el campo de texto submite el formulario utilizando el método submit del objeto form.

Control de campos de texto con Javascript

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.

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.

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](#).

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](#).

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 asteriscos en lugar del texto.

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.

Control de Checkbox en Javascript

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

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.

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](#).

Control de botones de radio en Javascript

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 buttons 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.

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.

Métodos del objeto radio

Son los mismos que los que tenía el [objeto checkbox](#).

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 elementos 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 control de botones de radio podemos acceder a un taller relacionado: [Inhibir radio button con Javascript](#)

Control de campos select con Javascript

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.

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.

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>.

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>.

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 la segunda variable.

Podemos [ver el ejemplo en funcionamiento aquí](#).

Podemos ver un ejemplo más práctico sobre qué se puede hacer con un campo select en el reportaje de [cómo hacer un navegador desplegable con Javascript](#).

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.

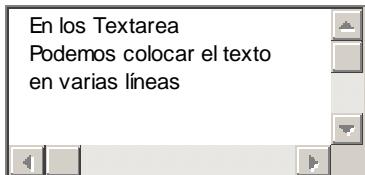
```
formul.miSelect.value
```

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.

Control de elementos Textarea en 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.



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 etiquetas <TEXTAREA> y su correspondiente cierre.

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.

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 de 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>

Los eventos 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.

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 manejador 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](#)

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

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

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

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 usuario 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 onmouseout 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

Ejemplos de eventos en Javascript. Onabort

A lo largo de los [manuales I](#) y [II](#) de Javascript, así como del [Taller](#), hemos mostrado muchos ejemplos de utilización de los manejadores de eventos. Aquí veremos ejemplos sencillos que se nos ocurren para utilizar otros manejadores 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 onmouseover 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)

Evento onabort

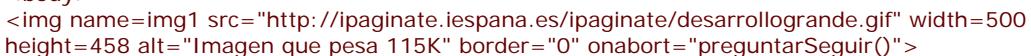
Veamos un primer ejemplo, en este caso sobre el evento onabort. Este evento se activa al cancelarse la carga de una página, ya sea porque se pulsa el botón de cancelar o porque el usuario se marcha de la página por otro enlace.

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/desarrollo grande.gif"
}
</script>
```

```

</head>
<body>

<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](#).

Ejemplo del evento onblur en Javascript

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 ha 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 vacía
                return ""
            }
        }
    </script>
</head>
<body>
    <input type="text" onblur="validarEntero(this.value)">
</body>

```

```

}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()
        //colojo 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.

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

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()
            //colocho 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()
            //colocho 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 caracteres 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](#).

Elementos de formulario select asociados

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.

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](#).

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>Selecione...
<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 opción 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 país elegido
  var pais
  pais = document.f1.pais[document.f1.pais.selectedIndex].value
  //miro a ver si el país 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
  }
}

```

```

//colojo 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.](#)

Evento onunload de Javascript

Veamos un ejemplo del evento onunload, que, recordamos, se activa cuando el usuario ha abandonado 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, abriendose 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()">

    <a href="http://www.desarrolloweb.com">DW!!</a>
</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.](#)

Evento onload de Javascript

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.

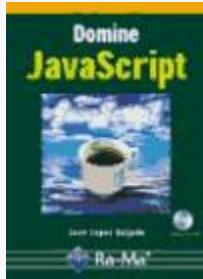


Motor Google

Motor DW

Portada | Monotemáticos | Secciones | Desarrolladores | Comunidad | Servicios gratuitos | Servicios profesionales

Inicio > Artículos

Manuales relacionados[+ Taller de Javascript](#)**Categorías**[+ Javascript](#)[+ Scripts en Javascript](#)**Libro recomendado****DOMINE JAVASCRIPT**Compra este libro en
Agapea, la librería urgente a
domicilio.

Inhibir un campo texto de formulario con Javascript

Hacer que un campo de formulario quede deshabilitado, es decir, que no se pueda posarse encima y cambiar su valor.

Esta vez toca un taller muy rápido y sencillo con Javascript para hacer que un campo de formulario de tipo texto se encuentre inhibido, es decir, que no podamos colocarnos encima de él para editar su contenido.

Referencia: Si lo que queremos es inhibir un campo de formulario de tipo radio (radio button) será necesaria otra técnica relatada en un taller distinto: [Inhibir radio button con Javascript](#)

Focus y Blur

La manera de hacerlo requiere el conocimiento de dos conceptos habituales de Javascript relacionados con el foco de la aplicación.

El concepto focus, está relacionado con ganar foco de la aplicación. El método **focus()**, que tienen los campos de texto y otros elementos de formulario, sirve otorgar el foco de la aplicación a ese elemento. El manejador de evento **onfocus** salta cuando un elemento gana el foco de la aplicación.

El concepto blur, está asociado a perder el foco de la aplicación. El método **blur()** sirve para que los elementos de formulario pierdan el foco y el manejador de eventos **onblur** se activa cuando el elemento al que lo apliquemos pierda el foco de la aplicación.

El ejercicio

Para inhibir un campo de formulario podemos hacer que el usuario nunca se pueda posar en ese elemento o bien, que si se llega a posar, se expulse inmediatamente. Para esto lo único que tememos que hacer es retirar el foco de un elemento cuando lo haya ganado

Nosotros utilizaremos el evento **onfocus** para detectar el instante en el que el elemento gana el foco y en ese momento haremos uso del método **blur()** para retirar el foco.

El código es extremadamente simple para tanta explicación:

```
<form>
<input type="text" value="122" onfocus="this.blur()">
</form>
```

El único detalle que merece la pena señalar es el uso de la palabra **this**, que hace referencia al elemento donde se está utilizando, en ese caso el campo de texto. **this.blur()** sería una simple llamada al método **blur()** en el elemento de formulario donde está colocada.

Puede verse en funcionamiento aquí:

Informe de [Miguel Angel Alvarez*](#)

Director de DesarrolloWeb.com

* Para consultas técnicas utilizar la [lista de correo](#).

Webs parecidas
[Maestrosdelweb](#)
[WebEstilo](#)
[Ciberteca](#)
[WebTaller](#)



[Versión imprimible del artículo](#)

[Enviar artículo por e-mail](#)

[Publicar un comentario del artículo](#)

Comentarios de los visitantes

Se muestra un comentario revisado

Comentario de eoes

11/12/03

Tambien se puede hacer sin utilizar scripts solo con readonly. Ejemplo:

```
<input name="campo" readonly type="text" id="campo" value="valor">
```

[Ir arriba](#)