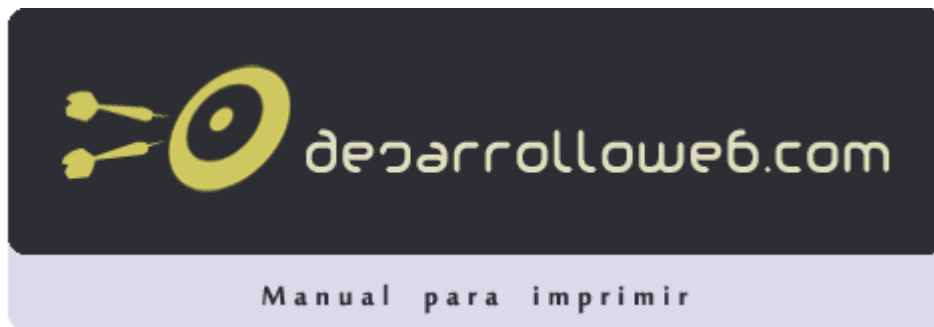


# Programación en JavaScript



## Autores del manual

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

**Miguel Angel Alvarez**

Director de DesarrolloWeb.com  
<http://www.desarrolloweb.com>  
(36 capítulos)

**Manu Gutierrez**

<http://www.tufuncion.com>  
(1 capítulo)

## **Introducción a Javascript**

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, cambien 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.

*Artículo por **Miguel Angel Alvarez***

## **Algo de historia**

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 un 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.

*Artículo por **Miguel Angel Alvarez***

## **Diferencias entre Java y Javascript**

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.

Artículo por *Miguel Angel Alvarez*

## Antes de empezar

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 [Cross Browser](#), que tiene una barra lateral que utiliza Javascript para dar dinamismo. También, sin ir más lejos, DesarrolloWeb.com utiliza Javascript para el menú superior, que muestra diferentes enlaces dentro de cada opción principal.

Por ejemplo, otra página que utiliza Javascript es [www.dynarch.com/](http://www.dynarch.com/), que incorpora menús dinámicos y algunas aplicaciones prácticas como un calendario.

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. Se pueden ver 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 desplegados, apertura de ventanas secundarias, validación de

formularios, 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 [DesarrolloWeb](http://DesarrolloWeb) 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](http://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](http://Home Site) o [UltraEdit](http://UltraEdit).

*Artículo por [Miguel Angel Alvarez](http://Miguel Angel Alvarez)*

## Versiones de navegadores y de Javascript

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).

*Artículo por [Miguel Angel Alvarez](#)*

## Efectos rápidos con Javascript

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).

Atrás

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.

Artículo por *Miguel Angel Alvarez*

## El lenguaje Javascript

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.

*Artículo por Miguel Angel Alvarez*

## **Maneras de ejecutar scripts**

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 el, debíamos indicarlo 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.

*Artículo por Miguel Angel Alvarez*

## Ocultar scripts en navegadores antiguos

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 hablemos 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 pusiesemos 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 javascript
</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>
```

Artículo por ***Miguel Angel Alvarez***

## Más sobre colocar scripts

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 language 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>.

Artículo por ***Miguel Angel Alvarez***

## Sintaxis Javascript

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.

*Artículo por [Miguel Angel Alvarez](#)*

## Variables 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
paísDeNacimiento
_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
```

*Artículo por Miguel Angel Alvarez*

## **Ambito de las variables 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 podremos 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. En resumen, la variable que tendrá validez en cualquier sitio de la página es la global. Menos en el ámbito donde está declarada la variable local, que será ella quien tenga 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 sobre escribir 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 `numero`, 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.

*Artículo por Miguel Angel Alvarez*

## Qué podemos guardar en variables

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 (").

### Boleanos

También contamos con el tipo booleano, que guarda una información que puede valer sí (`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.

*Artículo por **Miguel Angel Alvarez***

## ***Tipos de datos en Javascript***

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 el 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 cerrarían 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.

*Artículo por Miguel Angel Alvarez*

## Operadores Javascript I

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 la 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
```

Artículo por ***Miguel Angel Alvarez***

## Operadores Javascript II

### 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 = "hola"  
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 operandos 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
```

```
miBoleano = !miBoleano //miBoleano 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 numero 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
```

*Artículo por **Miguel Angel Alvarez***

## Operadores Javascript III

### 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 caso 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.

*Artículo por [Miguel Angel Alvarez](#)*

## Control de tipos

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

*Artículo por **Miguel Angel Alvarez***

## Estructuras de control

Los scripts vistos hasta ahora han sido tremendamente 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.

*Artículo por **Miguel Angel Alvarez***

## Estructura 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 el 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.

*Artículo por [Miguel Angel Alvarez](#)*

## Estructura IF (parte II)

### 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 también opcionales 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 mayor 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.

*Artículo por **Miguel Angel Alvarez***

## **Estructura SWITCH**

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.

Artículo por [Miguel Angel Alvarez](#)

## Bucle FOR

**El bucle FOR se utiliza para repetir mas 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 (inicialización;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](#).

*Artículo por Miguel Angel Alvarez*

## Bucles WHILE y DO WHILE

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](#).

*Artículo por **Miguel Angel Alvarez***

## **Break y continue**

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
  i++
}
```

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 hayen 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](#).

*Artículo por [Miguel Angel Alvarez](#)*

## Bucles anidados en Javascript

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 utilizareis 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 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.

*Artículo por [Miguel Angel Alvarez](#)*

## Funciones en Javascript

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 conozcan el concepto de función también será un capítulo util, 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.

```
NombreDeLaFuncion()
```

*Artículo por [Miguel Angel Alvarez](#)*

## Dónde colocamos las funciones

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>
```

Artículo por **Miguel Angel Alvarez**

## Parámetros de las funciones

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 realice 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. Esa 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, boleano 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.

*Artículo por Miguel Angel Alvarez*

## Valores de retorno

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 de el 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.

*Artículo por [Miguel Angel Alvarez](#)*

## Arrays en Javascript

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 compartimentos donde podemos introducir datos distintos. Para ello utilizamos un índice que nos permite especificar el compartimento 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 javascript

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 javascript 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 javascript**, 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 caracter.

```
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
```

*Artículo por **Miguel Angel Alvarez***

## Longitud de los arrays

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](#).

*Artículo por [Miguel Angel Alvarez](#)*

## Arrays multidimensionales

Los arrays multidimensionales son un 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 numero 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 día 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.

*Artículo por Miguel Angel Alvarez*

## ***Pausa y consejos Javascript***

Hasta aquí hemos visto la mayor parte de la sintaxis y forma de funcionar de el 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). Dale 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 boleanas 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.

*Artículo por **Miguel Angel Alvarez***

## **Tratamiento de errores en javascript**

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 icono 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](#).

*Artículo por **Miguel Angel Alvarez***

## Consejos para escribir código Javascript

Si estas dando tus primeros pasos en Javascript y estas empezando ya a ser sucio y desordenado... No tienes excusa da un giro para escribir el código ordenado y todo será más sencillo.

Los foros estan llenos de peticiones de información sobre Ajax, DOM y como se usan algunas librerías o efectos. Hay una extraordinaria cantidad de información, scripts, librerías que estan siendo desarrollados, blogs y nuevos sitios especializados en esta temática, sólo necesitas un poco de tiempo y echarle un vistazo... es muy fácil los mejores los encuentras en [Digg.com](http://Digg.com) o en [del.icio.us](http://del.icio.us), se acabaron aquellos días en el que Javascript y el DHTML se convirtieron en persona non grata como habilidad principal en tu CV.

La gran mayoría de código Javascript es hoy en dia mucho más limpio que en la "era" DHTML.

Ahora es un buen momento para convertirte en un entusiasta de Javascript. Aunque algunos defectos que ocurrieron tiempo atras se repiten sin embargo.

Aquí os dejo una series de consejos que os hará más sencillo mantener tu código Javascript ordenado, algunos consejos son demasiado obvios pero todos sabemos que el hombre es el único animal que...

### **Conserva la sintaxis y estructura de tu código limpia y ordenada**

Esto significa que guardes por ejemplo un límite de longitud de línea (80 caracteres) y que programes funciones razonablemente pequeñas. Un fallo es pensar que en una función larga lo podemos hacer todo.

Tener un tamaño razonable para tus funciones significa que las podrás reutilizar cuando

amplíes el código, tampoco seas extremista y haga funciones de una o dos líneas esto puede llegar a ser más confuso que usar una única función.

Este es un ejemplo que muestra cual es la justa medida en cuanto al tamaño de las funciones y la división de las tareas:

```
function toolLinks(){
var tools = document.createElement('ul');
var item = document.createElement('li');
var itemlink = document.createElement('a');
itemlink.setAttribute('href', '#');
itemlink.appendChild(document.createTextNode('close'));
itemlink.onclick = function(){window.close();}
item.appendChild(itemlink);
tools.appendChild(item);
var item2 = document.createElement('li');
var itemlink2 = document.createElement('a');
itemlink2.setAttribute('href', '#');
itemlink2.appendChild(document.createTextNode('print'));
itemlink2.onclick = function(){window.print();}
item2.appendChild(itemlink2);
tools.appendChild(item2);
document.body.appendChild(tools);
}
```

Puedes optimizar esta función separando cada tarea con su propia función:

```
function toolLinks(){
var tools = document.createElement('ul');
var item = document.createElement('li');
var itemlink = createLink('#', 'close', closeWindow);
item.appendChild(itemlink);
tools.appendChild(item);
var item2 = document.createElement('li');
var itemlink2 = createLink('#', 'print', printWindow);
item2.appendChild(itemlink2);
tools.appendChild(item2);
document.body.appendChild(tools);
}
```

```
function printWindow(){
window.print();
}
```

```
function closeWindow() {
window.close();
}
```

```
function createLink(url,text,func){
var temp = document.createElement('a');
temp.setAttribute('href', url);
temp.appendChild(document.createTextNode(text));
temp.onclick = func;
return temp;
}
```

## Utiliza inteligentemente los nombres de variables y funciones

Esta es una técnica esencial de programación, utiliza nombres de variables y funciones que sean totalmente descriptivos e incluso otra persona pueda llegar a plantearse que función realizan antes de ver el código.

Recuerda que es correcto el uso de guiones o mayúsculas para concatenar diferentes palabras, en este caso concreto de es más típico el uso de mayúsculas debido a la sintaxis del lenguaje, (ej. `getElementsByTagName()`).

```
CambioFormatoFecha();
cambio_formato_fecha();
```

## Comenta el código

Gracias a los comentarios puedes librarte de más de un quebradero de cabeza, es mejor resolver el problema una única vez.

Cómo puedes comprobar en cualquier libro de programación cada línea tiene comentarios explicando su objetivo.

## Diferencia las variables dependiendo de su importancia

Este paso es simple: Coloca aquellas variables que son usadas durante todo el script en la cabecera del código, de esta manera siempre sabrás donde encontrar estas variables que son las que determinan el resultado de nuestro código.

```
function toolLinks(){
var tools, closeWinItem, closeWinLink, printWinItem, printWinLink;

// variables temporales
var printLinkLabel = ?print?;
var closeLinkLabel = ?close?;#

tools = document.createElement(?ul?);
closeWinItem = document.createElement(?li?);
closeWinLink = createLink(?#, closeLinkLabel, closeWindow);
closeWinItem.appendChild(closeWinLink);
tools.appendChild(closeWinItem);
printWinItem = document.createElement(?li?);
printWinLink = createLink(?#, printLinkLabel, printWindow);
printWinItem.appendChild(printWinLink);
tools.appendChild(printWinItem);
document.body.appendChild(tools);
}
```

## Separa el texto del código

Puedes separar el texto del código, utilizando un documento llamado `texto.js` en formato JSON.

Un ejemplo que funciona muy bien podría ser:

```
var locales = {
'en': {
'homePageAnswerLink': 'Answer a question now',
'homePageQuestionLink': 'Ask a question now',
'contactHoverMessage': 'Click to send this info as a message',
'loadingMessage' : 'Loading your data...',
'noQAMessage' : 'You have no Questions or Answers yet',
'questionsDefault': 'You have not asked any questions yet',
'answersDefault': 'You have not answered any questions yet.',
'questionHeading' : 'My Questions',
'answersHeading' : 'My Answers',
'seeAllAnswers' : 'See all your Answers',
```

```
'seeAllQuestions' : 'See all your Questions',
'refresh': 'refresh'
},
'es': {
'homePageAnswerLink':'Responde una pregunta',
'homePageQuestionLink':'Haz una pregunta',
'contactHove' : 'Cargando datos...',
'noQAMessage' : 'No quedan preguntas',
'questionsDefault': 'Quedan preguntas por responder',
'answersDefault': 'No quedan preguntas pendientes',
'questionHeading' : 'Mis preguntas',
'answersHeading' : 'Mis respuestas',
'seeAllAnswers' : 'Ver todas las respuestas',
'seeAllQuestions' : 'Ver todas las preguntas',
'refresh': 'Refrescar'
},
'fr': {
}
'de': {
}
};
```

Esto permitiría a cualquiera que no es programador traducir el texto del script, cambiando únicamente las etiquetas sin necesidad de acceder al código.

## Documenta el código

Escribe una buena documentación de tu script / librería o efecto. Una buena documentación da calidad al código, sino pregúntate porque existe la clásica documentación en cualquier API con todas las posibles propiedades y parámetros, pero sin duda lo mejor de todo es explicar con ejemplos que contienen una lista de posibilidades.

*Artículo por **Manu Gutierrez***