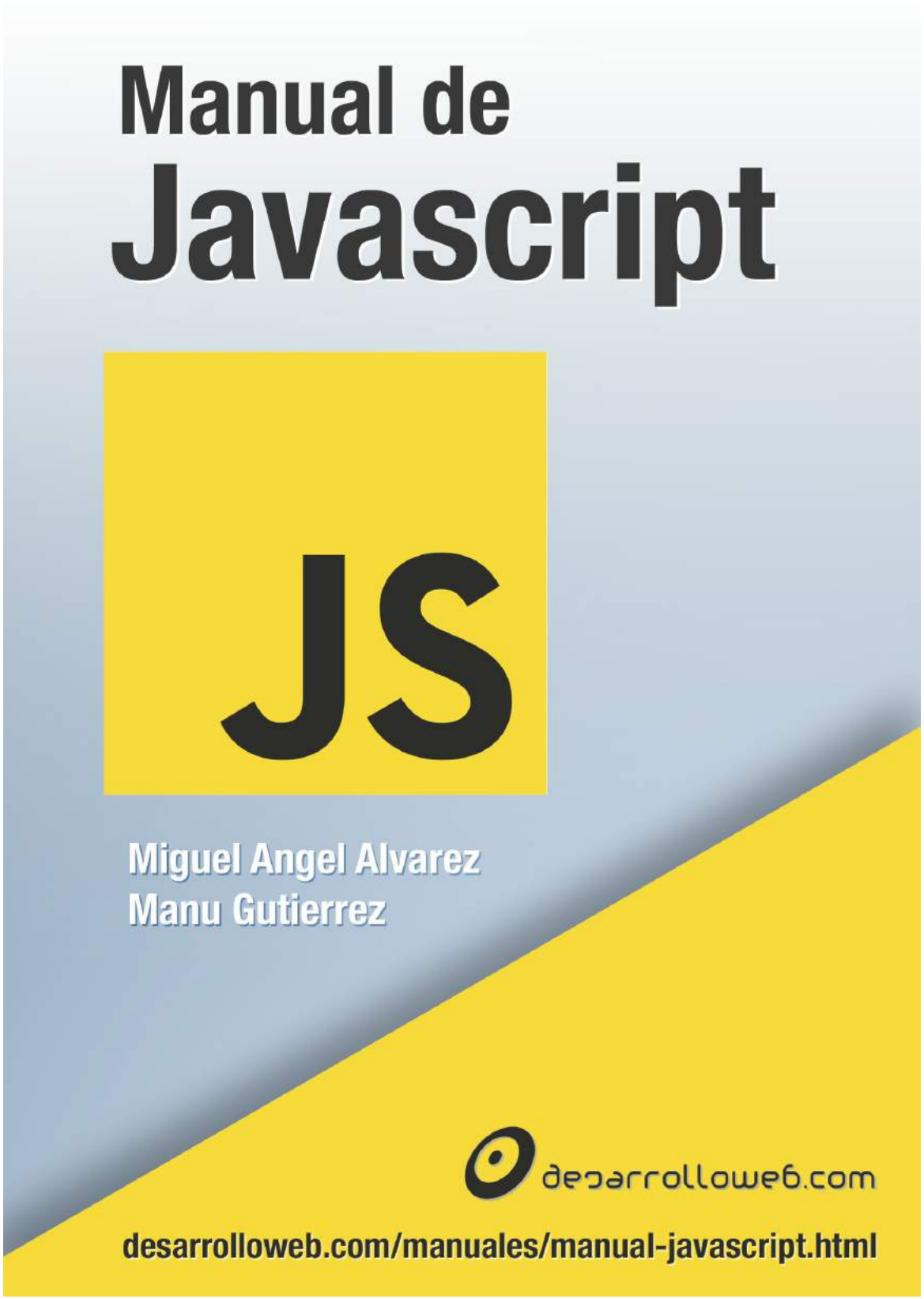
Manual de
Javascript
JS
Miguel Angel Alvarez
Manu Gutierrez
desarrolloweb.com
desarrolloweb.com/manuales/manual-javascript.html

Introducción: Manual de JavaScript

Manual de Javascript desde cero. Adéntrate en el lenguaje de programación más popular de la web y conoce todas las características de la programación del lado del cliente.

Javascript es el lenguaje de programación usado para las páginas web, compatible con todos los navegadores y que forma un estándar de desarrollo que ahora también se extiende a dispositivos o programas de propósito general multiplataforma.

Este manual de Javascript está dividido en dos grandes partes. En la primera parte veremos las características fundamentales del lenguaje, como su sintaxis, variables, estructuras de control, funciones, arrays, etc. Toda la información que encontrarás en la primera parte te sirve para programar Javascript a nivel general, sea donde sea el entorno de ejecución de tus programas.

En la segunda parte, que también encontrarás en esta misma página, ahondaremos en el uso de Javascript en el ámbito del navegador, es decir, aprenderemos a manejar los recursos que nos ofrece el navegador para hacer páginas interactivas, capaces de interactuar con el usuario de manera avanzada.

Encuentras este manual online en:
http://desarrolloweb.com/manuales/manual-javascript.html

Autores del manual

Las siguientes personas han participado como autores escribiendo artículos de este manual.

Miguel Angel Alvarez

Miguel es fundador de DesarrolloWeb.com y la plataforma de formación online EscuelaIT. Comenzó en el mundo del desarrollo web en el año 1997, transformando su hobby en su trabajo.

Manu Gutierrez

Introducciones a Javascript

Artículos que nos servirán para introducirnos en este lenguaje, aprendiendo los conceptos más básicos de Javascript y la programación del lado del cliente.

Introducción a Javascript

Veamos qué es JavaScript y las posibilidades que nos ofrece utilizar este lenguaje a la hora de desarrollar páginas web. Conocemos algo de su historia.

Con este artículo comenzamos el Manual de Javascript, en el que esperamos ofrecer un punto de partida para las personas que quieren introducirse en el mundo de la programación web del lado del cliente.

El curso de Javascript que hemos preparado en DesarrolloWeb.com está dividido en dos partes fundamentales. En este primer manual pretendemos explicar el lenguaje Javascript de manera general, ofreciendo información sobre cómo incluir scripts y lidiar con los elementos más básicos de cualquier lenguaje de programación, como son las variables, operadores, estructuras de control, funciones, etc. La segunda parte del manual la dedicaremos a explorar temas más específicos sobre cómo Javascript nos puede ayudar a aplicar dinamismo a una página web, a través del control dinámico de elementos de la página y la interacción con el usuario.

> **Nota:** Por tanto, en este primer manual encontraréis que se ofrece mucha más información teórica típica sobre lenguajes de programación. Será esencial para saber cómo dar los primeros pasos en la programación, pero quizás resulte un poco más aburrida que la segunda parte, donde aprenderemos a alterar dinámicamente la página web, responder a acciones del usuario, etc.
>
> Nosotros hemos querido explicar las cosas con detenimiento, para que aprender Javascript con este manual esté al alcance de personas incluso sin conocimientos de programación. No obstante, en DesarrolloWeb.com existen diversos manuales más básicos todavía para aprender a programar, como puede ser el curso de programación en vídeo, o más específicos para la web con el manual de Páginas dinámicas o la introducción a la programación.
>
> Sin embargo, quizás personas más experimentadas puedan preferir pasar directamente a la segunda parte de este manual, donde explicaremos cosas más prácticas y volver sobre artículos puntuales de este manual para utilizarlos como referencia a medida que vayan necesitando conocer la sintaxis de determinadas estructuras de control, operadores del lenguaje, construcción de funciones, etc.

En este artículo pretendemos explicar qué es Javascript y para qué sirve este lenguaje, al menos en líneas generales. A lo largo de éste y los próximos artículos explicaremos los conceptos necesarios para que las personas tengan una idea más o menos clara sobre Javascript, las posibilidades de del lenguaje o usos más comunes y los modos de trabajo que podemos emplear para desarrollar nuestros propios scripts.

Nota: Otro recurso que queremos recomendar para aprender Javascript, especialmente indicado para las personas con menos experiencia, es el Videotutorial de Javascript. Por supuesto, tampoco nos queremos olvidar de los Talleres de Javascript, así como otros manuales más específicos que encontrarás en la sección Javascript a fondo.

Qué es 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 manual vamos a tratar de acercarnos a este lenguaje en profundidad y conocer todos sus secretos y métodos de trabajo. Al final del manual seremos capaces de controlar el flujo en nuestros programas Javascript y saber cómo colocar scripts para resolver distintas necesidades que podamos tener. Todo lo que veremos a continuación nos servirá de base para adentrarnos más adelante en el desarrollo de páginas enriquecidas del lado del cliente.

Algo de la historia de Javascript

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 de dimensiones gigantes de textos interrelacionados por medio de enlaces. Cada una de las unidades básicas donde podemos encontrar información son 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.

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. Al poco tiempo quedó patente que HTML no era suficiente para realizar todas las acciones que se pueden llegar a necesitar en una página web. En otras palabras, HTML se había quedado corto ya que sólo sirve para presentar el texto en un página, definir su estilo y poco más.

Al complicarse los sitios web, una de las primeras necesidades fue que las páginas respondiesen a algunas acciones del usuario, para desarrollar pequeñas funcionalidades más allá de los propios enlaces. El primer ayudante para cubrir las necesidades que estaban surgiendo fue Java, que es un lenguaje de propósito general, pero que había creado una manera de incrustar programas en páginas web. A través de la tecnología de los Applets, se podía crear pequeños programas que se ejecutaban en el navegador dentro de las propias páginas web, pero que tenían posibilidades similares 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 adentrase 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 otros clientes web como Internet Explorer a partir de la versión 3.0. Sin embargo, la compañía Microsoft nombró a este lenguaje como JScript y tenía ligeras diferencias con respecto a Javascript, algunas de las cuales perduran hasta el día de hoy.

Diferencias entre distintos navegadores

Como hemos dicho el Javascript de Netscape y el de Microsoft Internet Explorer tenía ligeras diferencias, pero es que también el propio lenguaje fue evolucionando a medida que los navegadores presentaban sus distintas versiones y a medida que las páginas web se hacían más dinámicas y más exigentes las necesidades de funcionalidades.

Las diferencias de funcionamiento de Javascript han marcado las historia del lenguaje y el modo en el que los desarrolladores se relacionan con él, debido a que estaban obligados a crear código que funcionase correctamente en diferentes plataformas y diferentes versiones de las mismas. A día de hoy, siguen habiendo

muchas diferencias y para solucionarlo han surgido muchos productos como los Frameworks Javascript, que ayudan a realizar funcionalidades avanzadas de DHTML sin tener que preocuparse en hacer versiones distintas de los scripts, para cada uno de los navegadores posibles del mercado.

A continuación seguiremos aprendiendo curiosidades del lenguaje y aclararemos que Java y Javascript son dos cosas distintas, en el artículo sobre las diferencias de Java y Javascript.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *16/07/2001*
Disponible online en *http://desarrolloweb.com/articulos/introduccion-javascript.html*

Diferencias entre Java y Javascript

Java y Javascript son dos productos distintos. Ponemos de manifiesto la diferencia entre estos dos lenguajes con un nombre similar.

Estamos contando diversos asuntos interesantes y curiosidades que sirven de introducción para el Manual de Javascript y queremos tratar una de las más típicas asociaciones que se se hacen al oir hablar de Javascript. Nos referimos a relacionarlo con otro lenguaje de programación, llamado Java, que no tiene mucho que ver. Realmente Javascript se llamó así porque Netscape, que estaba aliado a los creadores de Java en la época, quiso aprovechar el conocimiento y la percepción que las personas tenían del popular lenguaje. Con todo, se creó un producto que tenía ciertas similitudes, como la sintaxis del lenguaje o el nombre. Se hizo entender que era un hermano pequeño y orientado específicamente para hacer cosas en las páginas web, pero también se hizo caer a muchas personas en el error de pensar que son lo mismo.

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 varible 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.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *16/07/2001*
Disponible online en *http://desarrolloweb.com/articulos/20.php*

Antes de empezar

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

Hay varios puntos que queremos comentar como introducción en el Manual de Javascript y que podrás querer conocer antes de comenzar a programar. Primero sería bueno hacernos una idea más concreta de las posibles aplicaciones que podría tener el lenguaje y que se pueden encontrar en innumerables sitios web. Además también queremos comentar las herramientas y conocimientos previos que necesitamos para ponernos manos a la obra.

Usos de Javascript

Quizás a día de hoy sobra decir para qué sirve Javascript, pero veamos brevemente algunos usos de este lenguaje que podemos encontrar en el web para hacernos una idea de las posibilidades que tiene.

Sin ir más lejos, DesarrolloWeb.com utiliza Javascript para el menú superior, que muestra diferentes enlaces dentro de cada opción principal. Vamos cambiando la página cada cierto tiempo, pero en el diseño actual de este sitio web, elementos como el recuadro de "Login" también tienen su dinamismo con Javascript.

Actualmente casi todas las páginas un poco avanzadas utilizan Javascript, pues se ha vuelto una de las insignias de lo que se denomina la Web 2.0 y la experiencia enriquecida de usuario. Por ejemplo, webs tan populares como Facebook, Twitter o Youtube usan Javascript a raudales. Para ser más concretos, cuando en la red social apretamos un enlace para comentar algo, se muestra en la página un pequeño formulario que aparece como por arte de magia y luego se envía sin salirse de la propia página. También cuando votamos por un vídeo en Youtube o cuando se cuentan los caracteres que llevamos escritos en los mini-post de Twitter, se utiliza Javascript para realizar pequeñas funcionalidades que no es posible realizar con HTML sólo. En realidad se pueden ver ejemplos de Javascript dentro de cualquier página un poco compleja. Algunos que habremos visto en innumerables ocasiones son calendarios dinámicos para seleccionar fechas, calculadoras o convertidores de divisas, editores de texto enriquecido, navegadores dinámicos, etc.

Es mucho más habitual encontrar Javascript para realizar efectos simples sobre páginas web, o no tan simples, como pueden ser navegadores dinámicos, 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. Sin embargo, aparte de esos unos simples ejemplos, podemos encontrar dentro de Internet muchas aplicaciones que basan parte de su funcionamiento en Javascript, que hacen que una página web se convierta en un verdadero programa interactivo de gestión de cualquier recurso. Ejemplos claros son las aplicaciones de ofimatica online, como Google Docs, Office Online o Google Calendar.

Qué necesitas para trabajar con Javascript

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

> **Nota:** Usuarios que deseen herramientas más avanzadas pueden encontrar en Internet fácilmente programas similares en la sección de programas para desarrolladores.

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 Komodo Edit, Notepad++ o UltraEdit.

Conocimientos previos recomendables

Lo cierto es que no hace falta tener mucha base de conocimientos para ponerse a programar en Javascript. Lo más seguro es que si lees estas líneas ya sepas todo lo necesario para trabajar, puesto que ya habrás tenido alguna relación con el desarrollo de sitios web y habrás detectado que para hacer ciertas cosas te viene bien conocer un poco de Javascript.

No obstante, sería bueno tener un dominio avanzado de HTML, al menos el suficiente para escribir código en ese lenguaje sin tener que pensar qué es lo que estás haciendo. También será útil un conocimiento medio sobre CSS y quizás alguna experiencia previa sobre algún lenguaje de programación, aunque en este manual de DesarrolloWeb.com vamos a tratar de explicar Javascript incluso para personas que no hayan programado nunca.

En el siguiente artículo seguiremos con temas que sirven de introducción al lenguaje de scripting del lado del cliente viendo las algunas diferencias de Javascript que existen en las versiones de navegadores que han ido apareciendo.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *16/07/2001*
Disponible online en *http://desarrolloweb.com/articulos/20.php*

Versiones de navegadores y de Javascript

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

Para continuar con la introducción al lenguaje que estamos viendo en el Manual 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:

> **Actualizado:** En el momento de actualizar este artículo, podemos decir que no tenemos que preocuparnos mucho de las versiones de Javascript que puedan haber existido. Realmente cualquier navegador medianamente moderno tendrá ahora todas las funcionalidades de Javascript que vayamos a necesitar y sobre todo, las que podamos utilizar en nuestros primeros pasos con el lenguaje. No obstante puede venir bien conocer las primeras versiones de Javascript que comentamos en este artículo, a modo de curiosidad.

- **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 estandar.
- **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).

Es obvio que todavía, después de escribir estas líneas, se presentarán o habrán presentado muchas otras versiones de Javascript, pues, a medida que se van mejorando los navegadores y van saliendo versiones de HTML, surgen nuevas necesidades para programación de elementos dinámicos. No obstante, todo lo que vamos a aprender en este manual, incluso otros usos mucho más avanzados, ya está implementado en cualquier Javascript que existan en la actualidad.

En el siguiente artículo comenzaremos ya a mostrar pequeños códigos Javascript que servirán para hacer efectos simples en páginas web.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *16/07/2001*
Disponible online en *http://desarrolloweb.com/articulos/20.php*

Efectos rápidos con Javascript

En este último artículo de introducción a Javascript veremos algunos ejemplos de códigos sencillos de gran utilidad.

Antes de meternos de lleno en materia podemos ver una serie de efectos rápidos que se pueden programar con Javascript, lo que nos puede hacer una idea más clara de las capacidades y potencia del lenguaje. A continuación veremos varios ejemplos, que hemos destacado para esta introducción en el Manual de Javascript, por tener un mínimo de complejidad y aunque sean muy básicos, 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, un botón normal con el aspecto predeterminado que el navegador y sistema operativo que usas otorgue a los botones. A continuación tienes una imagen sobre cómo se vería el botón en mi sistema.

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. Como podréis imaginar, existirían muchas otras muestras sencillas de Javascript que nos reservamos para capítulos posteriores.

Si lo deseas, puedes ver cómo hemos desarrollado algunos de estos efectos rápidos Javascript paso por paso y en vídeo. En el siguiente artículo empezaremos ya a hablar del propio lenguaje de programación Javascript.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *16/07/2001*
Disponible online en *http://desarrolloweb.com/articulos/20.php*

Primeros pasos con el lenguaje Javascript

Comenzamos a aprender cosas que tienen que ver directamente con la programación en el lenguaje Javascript y la manera con la que se integra éste en una página web.

El lenguaje Javascript

Veamos cómo se escribe el código en Javascript y para ello comentaremos las primeras reglas para insertar scripts en páginas web.

En esta parte del manual sobre Javascript vamos a conocer la manera más básica de trabajar con el lenguaje. En este artículo daremos las primeras informaciones sobre cómo incluir scripts, mezclando el propio código Javascript con el HTML.

Luego veremos también cómo se debe colocar código para que nuestra web sea compatible con todos los navegadores, incluso aquellos que no soportan Javascript. 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. Es decir, el código Javascript, en la mayoría de los casos, se mezcla con el propio código HTML para generar la página.

Esto quiere decir que debemos aprender a mezclar los dos lenguajes de programación y rápidamente veremos que, para que estos dos lenguajes puedan convivir sin problemas entre ellos, 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.

La colocación de los scripts sí que importa

En una misma página podemos introducir varios scripts, cada uno que podría introducirse dentro de unas etiquetas `<SCRIPT>` distintas. La colocación de estos scripts no es indiferente. En un principio, con lo que sabemos hasta el momento y los scripts que hemos realizado de prueba, nos da un poco igual donde colocarlos, pero en determinados casos esta colocación sí 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 com más detenidamente estas dos maneras de escribir scripts, que tienen como diferencia principal el momento en que se ejecutan las sentencias.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *29/07/2001*
Disponible online en *http://desarrolloweb.com/articulos/20.php*

Maneras de ejecutar scripts Javascript

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

Hasta ahora en el Manual de Javascript ya hemos tenido la ocasión de probar algunos scripts sencillos, no obstante, todavía tenemos que aprender una de las bases para poder trabajar con el lenguaje y es aprender las dos maneras de ejecutar código Javascript. Existen **dos maneras fundamentales 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.

Explicaremos ahora cada una de estas formas de ejecución disponibles, pero para el que lo desee, recomendamos también ver el vídeo sobre Maneras de incluir y ejecutar scripts.

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 esta parte del Manual de Javascript. En la segunda parte del Manual de Javascript podremos aprender muchas cosas y entre ellas veremos con detalle el segundo modo de ejecución de scripts que vamos a relatar a continuación.

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 indicarlas dentro del atributo onclick del botón.

Comprobamos pues que se puede introducir código Javascript dentro de determinados atributos de las etiquetas HTML. Veremos más adelante este tipo de ejecución en profundidad y los tipos de eventos que existen. Pero para llegar a ello aun tenemos que aprender muchas otras cosas de Javascript. En el próximo artículo mostraremos cómo podemos ocultar el código Javascript para navegadores antiguos.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *29/07/2001*
Disponible online en *http://desarrolloweb.com/articulos/20.php*

Ocultar scripts Javascript en navegadores antiguos

No todos los navegadores son compatibles con Javascript, así que tenemos que aprender cómo hacer que los scripts no molesten en navegadores que no los entienden.

A lo largo de los anteriores capítulos del Manual de Javascript ya hemos visto que el lenguaje se implementó a partir de Netscape 2.0 e Internet Explorer 3.0. Incluso, para los que no lo sepan, está bien decir que hay navegadores que funcionan en sistemas operativos, donde sólo se puede visualizar texto y donde determinadas tecnologías y aplicaciones no están disponibles, como el uso de imágenes, fuentes tipográficas distintas o el propio Javascript.

Así pues, no todos los navegadores que puedan utilizar los usuarios que visiten nuestra página 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 y como consecuencia, presentan los scripts en el cuerpo del documento, 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 -->). Además, en este artículo veremos también cómo mostrar un mensaje que se vea sólo en los navegadores que no son compatibles con Javascript.

Actualizado: En el momento actual podemos decir que casi el 100% de los navegadores disponibles soportan Javascript, o por lo menos reconocen las etiquetas de script, por lo que, aunque esté desactivado, no mostrarán el texto de nuestros programas Javascript. Por ello, actualmente ya no es fundamental realizar la operación de ocultar el código de los scripts de la página para navegadores antiguos. No obstante, si queremos hacer una página totalmente correcta, convendrá aprender cómo se puede ocultar un script para que en ningún caso se muestre como texto en la página.

Ocultar el código Javascript con comentarios HTML

Veamos con un ejemplo de código donde se han utilizado comentarios HTML para ocultar Javascript, o mejo dicho, el código los scripts Javascript.

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

Mostrar un mensaje para navegadores antiguos con `<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>
```

En el siguiente artículo veremos algunos otros detalles sobre colocar scripts Javascript que se han quedado en el tintero.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *29/07/2001*
Disponible online en *http://desarrolloweb.com/articulos/20.php*

Más sobre colocar scripts

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

Seguimos ofreciendo capítulos del Manual de Javascript, con información útil para saber cómo utilizar el lenguaje. Así que ahora veremos un par de notas adicionales sobre cómo colocar scripts en páginas web.

En este artículo mostraremos uno de los atributos que se pueden indicar en la etiqueta de SCRIPT, que indica el lenguaje que vamos a utilizar. Pero sobre todo debemos prestar especial atención al caso de incluir archivos externos con Javascript, que sin duda es una manera de trabajar, que utilizaréis bastante a menudo.

Indicar el 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 lenguage se escribe con dos -g- y no con -g- y con -j- como en castellano.

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

Actualizado: Hoy ya ni merece la pena nombrar otros lenguajes de scripting. Javascript es el único lenguaje para crear scripts en páginas web aceptado por la industria. Luego, el atributo language es realmente innecesario. Sin embargo, estamos obligados a definir el atributo "type", que señalamos en el siguiente punto. A pesar que en HTML 4.01 transicional nos valide correctamente el atributo language, no validará si estamos haciendo HTML strict, con lo que no recomendamos usar "language" en ningún caso. En los ejemplos de DesarrolloWeb.com donde se utilizaba language, por favor eliminarlo, pues no validaría correctamente con versiones estándar del lenguaje HTML actual.

Uso del atributo "type":

Cuando colocamos una etiqueta SCRIPT debemos usar el atributo "type" para indicar que tipo de codificación de script estamos haciendo y el lenguaje utilizado.

```
<script type="text/javascript">
```

El atributo "type" es necesario para que valide correctamente tu documento en las versiones más actuales del HTML.

Incluir 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 type="text/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>.

Vistos estos otros usos interesantes que existen en Javascript y que debemos conocer para poder aprovechar las posibilidades de la tecnología, debemos haber aprendido todo lo esencial para empezar a hacer cosas más importantes. Así que en el próximo artículo empezaremos a repasar la sintaxis del lenguaje Javascript.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *29/07/2001*
Disponible online en *http://desarrolloweb.com/articulos/20.php*

Sintaxis Javascript

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

Por fin empezamos a ver código fuente de Javascript! Esperamos que se hayan asimilado todas las informaciones previas del Manual de Javascript, en las que hemos aprendido básicamente diversos modos de incluir scripts en páginas web. Hasta ahora todo lo que hemos visto en este manual puede haber parecido muy teórico, pero de aquí en adelante esperamos que os parezca más ameno por empezar a ver cosas más prácticas y relacionadas directamente con la programación.

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 en el código

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 mas 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, ya sea de sintaxis o de referencia indefinida.

Por poner un ejemplo, no es lo mismo la función alert() que la función Alert(). La primera muestra un texto en una caja de diálogo y la segunda (con la primera A mayúscula) simplemente no existe, a no ser que la definamos nosotros. Como se puede comprobar, para que la función la reconozca Javascript, se tiene que escribir toda en minúscula. Otro claro ejemplo lo veremos cuando tratemos con variables, puesto que los nombres que damos a las variables también son sensibles a las mayúsculas y minúsculas.

Por regla general, los nombres de las cosas en Javascript se escriben siempre 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. Por ejemplo document.bgColor (que es un lugar donde se guarda el color de fondo de la página web), se escribe con la "C" de color en mayúscula, por ser la primera letra de la segunda palabra. 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.

En el próximo artículo comenzaremos a hablaros sobre la creación de variables en Javascript.

Fue publicado por primera vez en *30/07/2001*
Disponible online en *http://desarrolloweb.com/articulos/20.php*

Trabajo con variables y tipos de datos en Javascript

Una de las cosas más fundamentales en cualquier lenguaje de programación son las variables y los tipos de datos. Veremos qué son y cómo se trabaja con ellos en Javascript.

Variables en Javascript

Vemos en términos generales qué es una variable, para qué sirve y cómo declarar variables en Javascript antes de usarlas.

Este es el primero de los artículos que vamos a dedicar a las variables en el Manual de Javascript. Veremos, si no lo sabemos ya, que las variables son uno de los elementos fundamentales a la hora de realizar los programas, en Javascript y en la mayoría de los lenguajes de programación existentes.

Así pues, en este artículo veremos cuál es el concepto de variable y aprenderemos a declararlas en Javascript.

Concepto de variable

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

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

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

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

Edad paisDeNacimiento _nombre

También hay que evitar utilizar nombres reservados como variables, por ejemplo no podremos llamar a nuestra variable palabras como return o for, que ya veremos que son utilizadas para estructuras del propio

lenguaje. Veamos ahora algunos nombres de variables que no está permitido utilizar:

12meses tu nombre return pe%pe

Declaración de variables en Javascript

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.

Javascript cuenta con la palabra "var" que utilizaremos cuando queramos declarar una o varias variables. Como es lógico, se utiliza esa palabra para definir la variable antes de utilizarla.

> **Nota:** Aunque Javascript no nos obligue a declarar explícitamente las variables, es aconsejable declararlas antes de utilizarlas y veremos en adelante que se trata también de una buena costumbre. Además, en sucesivos artículos veremos que en algunos casos especiales, no producirá exactamente los mismos resultados un script en el que hemos declarado una variable y otro en el que no lo hagamos.

```
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
```

Si lo deseas, puedes ver un ejemplo de página que declara variables Javascript.

En el siguiente artículo seguiremos aprendiendo cosas de variables y veremos uno de los conceptos más importantes que deberemos aprender sobre ellas, el ámbito de las variables.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *30/07/2001*
Disponible online en *http://desarrolloweb.com/articulos/20.php*

Ambito de las variables en Javascript

El ámbito de las variables en Javascript: qué son las variables locales y globales y cómo se trabaja con ellas en Javascript.

El ámbito de las variables es uno de los conceptos más importantes que deberemos conocer cuando trabajamos con variables, no sólo en Javascript, sino en la mayoría de los lenguajes de programación.

En el artículo anterior ya comenzamos a explicar qué son las variables y cómo declararlas. En este artículo del Manual de Javascript pretendemos explicar con detenimiento qué es este ámbito de las variables y ofrecer ejemplos para que se pueda entender bien.

Concepto de ámbito de variables

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**.

En Javascript no podremos acceder a variables que hayan sido definidas en otra página. Por tanto, la propia página donde se define es el ámbito más habitual de una variable y le llamaremos a este tipo de **variables globales** a la página. Veremos también se pueden hacer variables con ámbitos distintos del global, es decir, variables que declararemos y tendrán validez 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 declarar variables con var, o no declararlas

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 varible 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 sobreescribir 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 funcion será la misma variable global numero declarada fuera de la función. En una situación como esta, al ejecutar la función se sobreescribirá la variable numero y el dato que había antes de ejecutar la función se perderá.

En el próximo artículo continuaremos hablando de variables y mostraremos que en ellas se pueden guardar distintos tipos de información.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *24/08/2001*
Disponible online en *http://desarrolloweb.com/articulos/20.php*

Qué podemos guardar en variables

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

En el Manual de Javascript ya hemos hablado sobre las variables en varios artículos. Pero todavía nos quedan cosas por ver y en concreto mostraremos en este artículo que en una variable podemos guardar distintos tipos de datos.

En una variable podemos introducir varios tipos de información. Por ejemplo podríamos introducir simple 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 más habituales 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 boleano, que guarda una información que puede valer si (true) o no (false).

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

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

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

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

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

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

Veremos algunas cosas más referentes a los tipos de datos en el próximo artículo.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *24/08/2001*
Disponible online en *http://desarrolloweb.com/articulos/20.php*

Tipos de datos en Javascript

Vemos los tres tipos de datos que soporta Javascript: numerico, boleano y texto.

En el artículo anterior del Manual de Javascript ya empezamos a mostrar que en una variable podemos

almacenar distintos tipos de información. No obstante, todavía hay algunas cosas que queremos explicar sobre los distintos tipos de datos disponibles 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. Javascipt 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 boleano

El tipo bolean, 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 boleanas son true o false.

```
miBoleana = true
<br>
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.

Con esto ya hemos terminado de explicar todo lo que se debe conocer sobre las variables en Javascript y podemos comenzar con un tema nuevo que será el de operadores.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *24/08/2001*
Disponible online en *http://desarrolloweb.com/articulos/20.php*

Operadores en Javascript

Tratamos en diversos artículos los operadores. Ofreceremos explicaciones de todos los operadores que podremos encontrarnos en Javascript.

Operadores Javascript

Estudiamos lo que es un operador y para qué sirve. Vemos los operadores de Javascript, en diversas clasificaciones, aritméticos, asignación, comparación, condicionales, a nivel de bit y preferencia de operadores.

En el presente artículo del Manual de Javascript vamos a comenzar una serie de textos enfocados en explicar los diferentes operadores disponibles en este lenguaje de programación.

Al desarrollar programas en cualquier lenguaje se utilizan los operadores, que sirven para hacer los cálculos y operaciones necesarios para llevar a cabo tus objetivos. Hasta el menor de los programas imaginables necesita de los operadores para realizar cosas, ya que un programa que no realizase operaciones, sólo se limitaría a hacer siempre lo mismo.

Es el resultado de las operaciones lo que hace que un programa varíe su comportamiento según los datos que tenga para trabajar y nos ofrezca resultados que sean relevantes para el usuario que lo utilice. Existen operaciones más sencillas o complejas, que se pueden realizar con operandos de distintos tipos, como números o textos, veremos en este capítulo, y los siguientes, de manera detallada todos estos operadores disponibles en Javascript.

Ejemplos de uso de operadores

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

3 + 5

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

```
miVariable = 23 * 5
```

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

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

Operadores aritméticos

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

- Suma de dos valores
- Resta de dos valores, también puede utilizarse para cambiar el signo de un número si lo utilizamos con un solo operando -23
- Multiplicación de dos valores / División de dos valores % El resto de la división de dos números (3%2 devolvería 1, el resto de dividir 3 entre 2) ++ Incremento en una unidad, se utiliza con un solo operando -- Decremento en una unidad, utilizado con un solo operando

Ejemplos

```
precio = 128 //introduzco un 128 en la variable precio
unidades = 10 //otra asignación, luego veremos operadores de asignación
factura = precio * unidades //multiplico precio por unidades, obtengo el valor factura
resto = factura % 3 //obtengo el resto de dividir la variable factura por 3
precio++ //incrementa en una unidad el precio (ahora vale 129)
```

Operadores de asignación

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

= Asignación. Asigna la parte de la derecha del igual a la parte de la izquierda. A al derecha se colocan los valores finales y a la izquierda generalmente se coloca una variable donde queremos guardar el dato. += Asignación con suma. Realiza la suma de la parte de la derecha con la de la izquierda y guarda el resultado en la parte de la izquierda. -= Asignación con resta *= Asignación de la multiplicación /= Asignación de la división %= Se obtiene el resto y se asigna

Ejemplos

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

En el siguiente artículo seguiremos conociendo otros de los operadores de Javascript: Operadores de cadenas, operadores lógicos y operadores condicionales.

Operadores de cadenas

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

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

Ejemplo

```
cadena1 = "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 élla.

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

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

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

> **Nota:** Como se puede haber imaginado, faltan muchas operaciones típicas a realizar con cadenas, para las cuales no existen operadores. Es porque esas funcionalidades se obtienen a través de la clase String de Javascript, que veremos más adelante.

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 boleanos, que conocimos anteriormente, que son el verdadero (true) y el falso (false). Los operadores lógicos relacionan los operandos boleanos para dar como resultado otro operando boleano, 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 boleano 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.

Nota: Para no llevarnos a engaño, cabe decir que los operadores lógicos pueden utilizarse en combinación con tipos de datos distintos de los boleanos, pero en este caso debemos utilizarlos en expresiones que los conviertan en boleanos. En el siguiente grupo de operadores que vamos a tratar en este artículo hablaremos sobre los operadores condicionales, que se pueden utilizar junto con los operadores lógicos para realizar sentencias todo lo complejas que necesitemos. Por ejemplo:

```
if (x==2 && y!=3){
   //la variable x vale 2 y la variable y es distinta de tres
}
```

En la expresión condicional anterior estamos evaluando dos comprobaciones que se relacionan con un operador lógico. Por una parte x==2 devolverá un true en caso que la variable x valga 2 y por otra, y!=3 devolverá un true cuando la variable y tenga un valor distinto de 3. Ambas comprobaciones devuelven un boleano cada una, que luego se le aplica el operador lógico && para comprobar si ambas comprobaciones se cumplieron al mismo tiempo.

Sobra decir que, para ver ejemplos de operadores condicionales, necesitamos aprender estructuras de control como if, a las que no hemos llegado todavía.

! 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

```
miBoleano = 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.

Nota: Por supuesto, los operadores condicionales sirven también para realizar expresiones en las que se comparan otros tipos de datos. Nada impide comparar dos cadenas, para ver si son iguales o distintas,

por ejemplo. Incluso podríamos comparar boleanos.

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 valores son iguales != Comprueba si dos valores son distintos

Mayor que, devuelve true si el primer operando 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

Veremos ejemplos de operadores condicionales cuando expliquemos estructuras de control, como la condicional if.

De manera adicional, en este texto veremos un asunto de bastante importancia en la programación en general, que es la precedencia de operadores, que debemos tener en cuenta siempre que utilicemos diversos operadores en una misma expresión, para que se relacionen entre ellos y se resuelvan de la manera habíamos planeado.

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 boleanos = += -= *= /= %= <<= >>= >>>= &= ^= != 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
```

Lo que nos da como resultado el valor siguiente.

```
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.

Para acabar con el tema de operadores nos queda por ver un último operador un tanto especial, llamado typeof.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *20/09/2001*
Disponible online en *http://desarrolloweb.com/articulos/20.php*

Operadores Javascript II

Estudiamos los operadores de cadenas, lógicos y condicionales.

Este es el segundo artículo sobre los operadores que publicamos en el Manual de Javascript, pues son suficientes como para verlos por partes. En la anterior entrega ofrecimos una breve descripción del concepto de operador y vimos los operadores aritméticos y de asignación.

En el presente texto veremos un listado y descripción de tres tipos adicionales: operadores de cadenas, operadores lógicos y operadores condicionales.

Operadores de cadenas

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

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

Ejemplo

```
cadena1 = "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 élla.

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

```
miNumero = 23

miCadena1 = "pepe"

miCadena2 = "456"

resultado1 = miNumero + miCadena1 //resultado1 vale "23pepe"

resultado2 = miNumero + miCadena2 //resultado2 vale "23456"

miCadena2 += miNumero //miCadena2 ahora vale "45623"
```

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

Nota: Como se puede haber imaginado, faltan muchas operaciones típicas a realizar con cadenas, para las cuales no existen operadores. Es porque esas funcionalidades se obtienen a través de la clase String de Javascript, que veremos más adelante.

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 boleanos, que conocimos anteriormente, que son el verdadero (true) y el falso (false). Los operadores lógicos relacionan los operandos boleanos para dar como resultado otro operando boleano, 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 boleano 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.

Nota: Para no llevarnos a engaño, cabe decir que los operadores lógicos pueden utilizarse en combinación con tipos de datos distintos de los boleanos, pero en este caso debemos utilizarlos en expresiones que los conviertan en boleanos. En el siguiente grupo de operadores que vamos a tratar en este artículo hablaremos sobre los operadores condicionales, que se pueden utilizar junto con los operadores lógicos para realizar sentencias todo lo complejas que necesitemos. Por ejemplo:

```
if (x==2 && y!=3){
   //la variable x vale 2 y la variable y es distinta de tres
}
```

En la expresión condicional anterior estamos evaluando dos comprobaciones que se relacionan con un operador lógico. Por una parte x==2 devolverá un true en caso que la variable x valga 2 y por otra, y!=3 devolverá un true cuando la variable y tenga un valor distinto de 3. Ambas comprobaciones devuelven un boleano cada una, que luego se le aplica el operador lógico && para comprobar si ambas comprobaciones se cumplieron al mismo tiempo.

Sobra decir que, para ver ejemplos de operadores condicionales, necesitamos aprender estructuras de control como if, a las que no hemos llegado todavía.

! 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

```
miBoleano = 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.

> **Nota:** Por supuesto, los operadores condicionales sirven también para realizar expresiones en las que se comparan otros tipos de datos. Nada impide comparar dos cadenas, para ver si son iguales o distintas, por ejemplo. Incluso podríamos comparar boleanos.

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 valores son iguales != Comprueba si dos valores son distintos

> Mayor que, devuelve true si el primer operando 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

Veremos ejemplos de operadores condicionales cuando expliquemos estructuras de control, como la condicional if.

En el siguiente artículo finalizaremos el tema de operadores, explicando los operadores a nivel de bit y la precedencia de operadores.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *20/09/2001*
Disponible online en *http://desarrolloweb.com/articulos/20.php*

Operadores Javascript III

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

Continuamos con el tema de operadores, que ya habíamos iniciado en capítulos anteriores de este manual de Javascript. En estos momentos ya hemos visto casi todos los tipos de operadores y sólo nos queda conocer un grupo que realmente no se utiliza casi nunca, que son los operadores de nivel de bit.

De manera adicional, en este texto veremos un asunto de bastante importancia en la programación en general, que es la precedencia de operadores, que debemos tener en cuenta siempre que utilicemos diversos operadores en una misma expresión, para que se relacionen entre ellos y se resuelvan de la manera habíamos planeado.

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 boleanos = += -= *= /= %= <<= >>= >>>= &= ^= != 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
```

Lo que nos da como resultado el valor siguiente.

```
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.

Para acabar con el tema de operadores nos queda por ver un último operador un tanto especial, llamado typeof.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *20/09/2001*
Disponible online en *http://desarrolloweb.com/articulos/20.php*

Control de tipos

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

El listado de operadores que venimos ofreciendo para el Manual de Javascript se completa con el operador typeof, que veremos a continuación.

Hemos podido comprobar que, para determinados operadores, 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.

Por ejemplo, cuando utilizábamos el operador +, si se trataba de números los sumaba, pero si se trataba de cadenas de caracteres las 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.

> **Nota:** a lo largo de nuestra experiencia con Javascript veremos que muchas veces es más útil cambiar el tipo de dato de una variable antes de hacer una comprobación con tipeof para ver si la podemos utilizar como operando. Existen diversas funciones para intentar cambiar el tipo de una variable, como parseInt(), que veremos más adelante en la Segunda Parte del Manual de Javascript.

```
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)
```

Si ejecutamos este script obtendremos que en la página se escribirá el siguiente texto:

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 ser imprimen en la página los distintos tipos de las variables. Estos pueden ser los siguientes:

- **boolean**, para los datos boleanos. (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.

Con esto ya hemos terminado de ver la lista de operadores que podemos utilizar en Javascript y en el próximo artículo pasaremos a conocer las estructuras de control.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *03/10/2001*
Disponible online en *http://desarrolloweb.com/articulos/20.php*

Estructuras de control en Javascript

Las estructuras de control nos permitirán controlar el flujo de nuestros programas. Por supuesto, también forman parte de los asuntos más básicos de Javascript y de cualquier lenguaje de programación, por lo que las veremos con detenimiento.

Estructuras de control en Javascript

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

Los scripts vistos hasta ahora en el Manual de Javascript 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 y de hecho, en la mayoría de los casos las cosas son bastante más complejas.

Si tenemos alguna experiencia en la programación sabremos que en los programas generalmente se necesitará hacer cosas distintas dependiendo del estado de nuestras variables o realizar un mismo proceso muchas veces sin escribir las mismas líneas de código una y otra vez. Para realizar cosas más complejas en nuestros scripts se utilizan las estructuras de control. Con ellas 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. Comenzaremos con la estructura de control if.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *03/10/2001*
Disponible online en *http://desarrolloweb.com/articulos/20.php*

Estructura IF en Javascript

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

En el Manual de Javascript de DesarrolloWeb.com ya empezamos a explicar lo que son las estructuras de control. En el presente artículo vamos a dedicarnos a mostrar cómo funciona la sentencia if, que es la estructura más habitual de las utilizadas para tomar decisiones en programas informáticos.

IF es una estructura de control utilizada para **tomar decisiones**. Es un condicional que sirve para realizar 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.

Nota: Todas las estructuras de control se escriben en minúsculas en Javascript. Aunque algunas veces para destacar el nombre de la estructura la podamos escribir en el texto del manual con letras mayúsculas, en el código de nuestros scripts siempre tenemos que ponerlo en minúsculas. En caso contrario recibiremos un mensaje de error.

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

Opcionalmente se pueden indicar acciones a realizar en caso de que la evaluación de la sentencia devuelva

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.

Nota: Aunque las llaves para englobar las sentencias a ejecutar tanto en el caso positivo como negativo sean opcionales cuando queremos ejecutar una única sentencia, la recomendación es colocarlas siempre, porque obtendremos así un código fuente más claro. Por ejemplo:

```
if (llueve)
   alert("Cae agua");
```

Sería exactamente igual que este código:

```
if (llueve){
   alert("Cae agua");
}
```

O incluso, igual a este otro:

```
if (llueve) alert("Cae agua");
```

Sin embargo, cuando utilizamos las llaves, el código queda bastante más claro, porque se puede apreciar en un rápido vistazo qué instrucciones están dependiendo del caso positivo del if. Esto es un detalle que ahora quizás no tenga mucha importancia, pero que se agradecerá cuando el programa sea más complejo o cuando varios programadores se encarguen de tocar un mismo código.

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.

Nota: Nosotros, así como lo haría cualquier persona con cierta experiencia en el área de programación, aconsejamos que se utilicen los sangrados y saltos de línea necesarios para que las instrucciones se

puedan entender mejor. Quizás el día que realizas un código tengas claro qué has hecho y por qué es así, pero dentro de un mes, cuando tengas que releer ese código, quizás te acuerdes menos de lo que hiciste en tus scripts y agradecerás que tengan un formato amigable para que se puedan leer con facilidad por las personas. Si trabajas en equipo estas recomendaciones serán todavía más importantes, puesto que todavía es más complicado leer código fuente que han realizado otras personas.

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 (aunque sí sería recomendable haberlas puesto). 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 += nuevoArticulo //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 boleano. Por ejemplo un operador condicional es el operador "es igual" (==), que devuelve true en caso de que los dos operandos sean iguales o false en caso de que sean distintos.

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

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

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

```
if (bateria < 0.5 && 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 es menor que 0.5 (está casi 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 casi sin batería Y sin red eléctrica, informo que el ordenador se va a apagar.

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

La estructura if es de las más utilizadas en lenguajes de programación, para tomar decisiones en función de la evaluación de una sentencia. En el artículo anterior del Manual de Javascript ya comenzamos a explicar la estructura if y ahora vamos a ver algunos usos un poquito más avanzados.

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 informando de ello. 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.

Nota: En este manual utilizaré una notación como la que has podido ver en las líneas anteriores. 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 y donde cuanto menos escribamos más elegantes seremos. 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 menor 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.

Para ampliar la información recomendamos ver también el Videotutorial de Javascript, en el vídeo donde tratamos la estructura IF.

Este artículo es obra de *Miguel Angel Alvarez*

Fue publicado por primera vez en *03/10/2001*
Disponible online en *http://desarrolloweb.com/articulos/20.php*

Estructura IF (parte II)

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

La estructura if es de las más utilizadas en lenguajes de programación, para tomar decisiones en función de la evaluación de una sentencia. En el artículo anterior del Manual de Javascript ya comenzamos a explicar la estructura if y ahora vamos a ver algunos usos un poquito más avanzados.

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 informando de ello. 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.

> **Nota:** En este manual utilizaré una notación como la que has podido ver en las líneas anteriores. 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 y donde cuanto menos escribamos más elegantes seremos. 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 menor 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.

En el Manual de Javascript de DesarrolloWeb.com ya empezamos a explicar lo que son las estructuras de control. En el presente artículo vamos a dedicarnos a mostrar cómo funciona la sentencia if, que es la estructura más habitual de las utilizadas para tomar decisiones en programas informáticos.

IF es una estructura de control utilizada para **tomar decisiones**. Es un condicional que sirve para realizar 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.

> **Nota:** Todas las estructuras de control se escriben en minúsculas en Javascript. Aunque algunas veces para destacar el nombre de la estructura la podamos escribir en el texto del manual con letras mayúsculas, en el código de nuestros scripts siempre tenemos que ponerlo en minúsculas. En caso contrario recibiremos un mensaje de error.

```
if (expresión) {

   //acciones a realizar en caso positivo

   //...

}
```

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

> **Nota:** Aunque las llaves para englobar las sentencias a ejecutar tanto en el caso positivo como negativo sean opcionales cuando queremos ejecutar una única sentencia, la recomendación es colocarlas siempre, porque obtendremos así un código fuente más claro. Por ejemplo:
>
> Veamos el código siguiente:
>
> ```
> if (llueve)
> ```

```
    alert("Cae agua");
```

Sería exactamente igual que este código:

```
if (llueve){
   alert("Cae agua");
}
```

O incluso, igual a este otro:

```
if (llueve) alert("Cae agua");
```

Sin embargo, cuando utilizamos las llaves, el código queda bastante más claro, porque se puede apreciar en un rápido vistazo qué instrucciones están dependiendo del caso positivo del if. Esto es un detalle que ahora quizás no tenga mucha importancia, pero que se agradecerá cuando el programa sea más complejo o cuando varios programadores se encarguen de tocar un mismo código.

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.

Nota: Nosotros, así como lo haría cualquier persona con cierta experiencia en el área de programación, aconsejamos que se utilicen los sangrados y saltos de línea necesarios para que las instrucciones se puedan entender mejor. Quizás el día que realizas un código tengas claro qué has hecho y por qué es así, pero dentro de un mes, cuando tengas que releer ese código, quizás te acuerdes menos de lo que hiciste en tus scripts y agradecerás que tengan un formato amigable para que se puedan leer con facilidad por las personas. Si trabajas en equipo estas recomendaciones serán todavía más importantes, puesto que todavía es más complicado leer código fuente que han realizado otras personas.

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 (aunque sí sería recomendable haberlas puesto). 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 += nuevoArticulo //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 boleano. Por ejemplo un operador condicional es el operador "es igual" (==), que devuelve true en caso de que los dos operandos sean iguales o false en caso de que sean distintos.

```
if (edad > 18)

   document.write("puedes ver esta página para adultos")
```

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

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

```
if (bateria < 0.5 && 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 es menor que 0.5 (está casi

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 casi sin batería Y sin red eléctrica, informo que el ordenador se va a apagar.

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

En el siguiente artículo seguiremos explicando usos avanzados de la estructura de control if en Javascript, como la anidación de estructuras if. Así mismo, recomendamos ver también el Videotutorial de Javascript, en el vídeo donde tratamos la estructura IF.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *03/10/2001*
Disponible online en *http://desarrolloweb.com/articulos/20.php*

Estructura SWITCH de Javascript

La estructura de control switch de Javascript es utilizada para tomar decisiones en función de distintos estados o valores de una variable.

Las estructuras de control son la manera con la que se puede dominar el flujo de los programas, para hacer cosas distintas en función de los estados de las variables. En el Manual de Javascript ya empezamos a ver las estructuras de control y ahora le ha tocado el turno a SWITCH, una estructura un poco más compleja que permite hacer múltiples operaciones dependiendo del estado de una variable.

En este artículo veremos que switch nos sirve 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.

Nota: Además contamos con un videotutorial sobre el SWITCH en Javascript que os puede ser de mucha ayuda para entenderlo todo mucho mejor.

En el siguiente artículo comenzaremos a explorar las estructuras de control para hacer bucles o repeticiones en Javascript, comenzando por el bucle for.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *03/10/2001*
Disponible online en *http://desarrolloweb.com/articulos/20.php*

Bucle FOR en Javascript

Descripción y ejemplos de funcionamiento del bucle FOR.

Comenzamos con este artículo del Manual de Javascript a explorar las estructuras de control para producir bucles o repeticiones, comenzando por el bucle for, no por ser el más simple, pero sí el más utilizado en la programación.

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

```
for (inicialización; condición; actualización) {
      //sentencias a ejecutar en cada iteración
}
```

El bucle FOR tiene tres partes incluidas entre los paréntesis, que nos sirven para definir cómo deseamos que se realicen las repeticiones. La primera parte 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 una iteración del bucle. Contiene una expresión para decidir 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)
	document.write("<br>")
}
```

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**, lo que permite una rápida configuración del bucle y una versatilidad enorme.

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.

> **Nota:** 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, comenzando en el valor 343 y siempre que la variable tenga un valor mayor o igual que 10.

Ejercicio de ejemplo del bucle for

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.

Ahora que ya conocemos el bucle for, estamos en condiciones de aprender a manejar otras estructuras de control para realizar repeticiones, como los bucles while y do...while.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *18/10/2001*
Disponible online en *http://desarrolloweb.com/articulos/bucle-for-javascript.html*

Bucles WHILE y DO WHILE

Descripción y diferentes usos de los dos tipos de bucles WHILE que se encuentran disponibles en Javascript, con algunos ejemplos prácticos.

Estamos tratando acerca de las distintas estructuras de control que existen en el lenguaje Javascript y en concreto viendo los distintos tipos de bucles que podemos implementar en este lenguaje de programación. En artículos anteriores del Manual de Javascript vimos ya el primero de los bucles que debemos conocer, el bucle for y ahora vamos a tratar sobre los otros dos tipos de estructuras de control para hacer repeticiones. Así pues, 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 = prompt("dame un color (escribe rojo para salir)","")
}
```

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 y lo hace repetidas veces, mientras que el color introducido 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 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".

> **Nota:** Hemos utilizado en este ejemplo la función prompt de Javascript, que no hemos visto todavía en este manual. Esta función sirve para que mostrar una caja de diálogo donde el usuario debe escribir un texto. Esta función pertenece al objeto window de Javascript y la comentamos en el artículo Métodos de window en Javascript.

Bucle DO...WHILE

El bucle do...while es la última de las estructuras para implementar repeticiones de las que dispone en Javascript y es una variación del bucle while visto anteriormente. 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 cualquiuer 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 = prompt("dame un color (escribe rojo para salir)","")
} 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 de uso de los bucles while

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 (dependerá de los valores aleatorios que se vayan obteniendo).

```
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. De todos modos, si lo deseas, puedes ver este artículo que habla sobre números aleatorios en Javascript.

Podemos ver una página con el ejemplo en funcionamiento.

Con esto ya hemos conocido todos los tipos de bucles que existen en Javascript, no obstante aun vamos a dedicar un artículo para explicar las sentencias break y continue que nos sirven para alterar el funcionamiento normal de los bucles en dos sentidos.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *18/10/2001*
Disponible online en *http://desarrolloweb.com/articulos/20.php*

Break y continue

Dos instrucciones que aumentan el control sobre los bucles en Javascript. Sirven para parar y continuar con la siguiente iteración del bucle respectivamente.

Javascript tiene diferentes estructuras de control para implementar bucles, como FOR, WHILE y DO...WHILE, que ya hemos podido explicar en capítulos anteriores del Manual de Javascript. Como hemos podido comprobar, con estos bucles podemos abarcar gran cantidad de necesidades, pero quizás con el tiempo encuentres que te faltan algunas posibilidades de control de las repeticiones de los bucles.

Imagina por ejemplo que estas haciendo un bucle muy largo para encontrar algo en cientos o miles de sitios. Pero ponte en el caso que durante las primeras iteraciones encuentres ese valor que buscabas. Entonces no tendría sentido continuar con el resto del bucle para buscar ese elemento, pues ya lo habías encontrado. En estas situaciones nos conviene saber para el bucle cancelar el resto de iteraciones. Obviamente, ésto es solo un ejemplo de cómo podríamos vernos en la necesidad de controlar un poco más el bucle. En la vida real como programador encontrarás muchas otras ocasiones en las que te interesará hacer esto u otras cosas con ellos.

Así pues, existen dos instrucciones que se pueden usar en de las distintas estructuras de control y principalmente en los bucles, que te servirán para controlar dos tipos de situaciones. **Son las instrucciones break y continue**.:

- **break:** Significa detener la ejecución de un bucle y salirse de él.
- **continue:** Sirve para detener la iteración actual y volver al principio del bucle para realizar otra iteración, si corresponde.

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 = prompt("dime si continuo preguntando...", "si")
       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", situación en la cual 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 = prompt("La cuenta está en " + i + ", dime si incremento", "si")
```

```
        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 ignorarían las sentencia que hayan por debajo del continue.

Ejemplo adicional de la sentencia break

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.

Con la descripción de las sentencias break y continue hemos podido abarcar todo lo que se debe saber sobre la creación de bucles en Javascript. Ahora bien, en el siguiente artículo todavía vamos a seguir en el tema de las estructuras de control, porque queremos ofrecer un ejemplo un poco más avanzado donde aprenderemos a anidar bucles.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *18/10/2001*
Disponible online en *http://desarrolloweb.com/articulos/20.php*

Bucles anidados en Javascript

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

En el Manual de Javascript hemos recorrido ya diversos artículos para hablar de bucles. En este momento no debería haber ningún problema para poder crear los distintos tipos de bucles sin problemas, no obstante, queremos dedicar un artículo completo a tratar acerca de uno de los usos más habituales de los bucles, que podremos encontrar cuando estemos haciendo programas más complejos: la anidación de bucles.

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. Si en vuestra experiencia como programadores los habéis anidado un bucle todavía, tener certeza

que más tarde o temprano os encontraréis con esa necesidad.

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.

Debido al flujo del programa en esquemas de anidación como el que hemos visto, 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.

Nota: 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.

Con este artículo más bien práctico sobre anidación de bucles, terminamos el tema de las estructuras de control. Ahora pasaremos a otra sección de este Manual de Javascript, en la que trataremos sobre las funciones.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *10/12/2001*
Disponible online en *http://desarrolloweb.com/articulos/20.php*

Funciones en Javascript

Las funciones nos permitirán hacer programas y scripts más optimizados y de fácil mantenimiento. También son básicas en cualquier lenguaje de programación y les dedicaremos varios artículos.

Funciones en Javascript

Comenzamos con las funciones en Javascript. Definimos el concepto de función y aprendemos a crearlas y a llamarlas.

Seguimos trabajando y ampliando nuestros conocimientos sobre Javascript. Con lo visto hasta ahora en el Manual de Javascript ya tenemos una cierta soltura para trabajar en este interesante lenguaje de programación. Pero todavía nos queda mucho por delante.

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 útil, pues también veremos la sintaxis y funcionamiento de las funciones en Javascript.

Qué es una función

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

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

> **Nota:** Si queremos, podemos ampliar esta descripción de las funciones en el artículo Concepto de función.

Las funciones se utilizan constantemente, no sólo las que escribes tú, sino también las que ya están definidas en el sistema, pues todos los lenguajes de programación suelen tener 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. Por ejemplo, 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 manual 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 reconozca 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. Admitimos que 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 muestran 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

Para ejecutar una función la tenemos que invocar 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. Por ejemplo, así llamaríamos a la función escribirBienvenida() que acabamos de crear.

```
escribirBienvenida()
```

Luego veremos que existen muchas cosas adicionales que debemos conocer de las funciones, como el paso de parámetros o los valores de retorno. Pero antes vamos a explicar dónde debemos colocar las funciones

Javascript.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *02/11/2001*
Disponible online en *http://desarrolloweb.com/articulos/20.php*

Dónde colocamos las funciones Javascript

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

Las funciones son uno de los principales componentes de los programas, en la mayoría de los lenguajes de programación. En el Manual de Javascript ya hemos comenzado a explicar qué es una función y cómo podemos crearla e invocarla en este lenguaje. Ahora vamos a tratar un tema que no es tanto de sintaxis y programación, sino que tiene más que ver con el uso correcto y habitual que se hace de las funciones en Javascript, que no es otro que la colocación del código de las funciones en la página web.

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

Existen dos opciones posibles para colocar el código de una función:

a) Colocar la función en el mismo bloque de script: En concreto, la función se puede definir en el bloque <SCRIPT> donde esté la llamada a la función, aunque es indiferente si la llamada se encuentra antes o después del código de 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.

b) Colocar la función en otro bloque de script: 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 tenemos funciones Javascript. Como se puede comprobar, las funciones están en la cabecera de la página (dentro del HEAD). Éste es un lugar excelente 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 sabemos seguro que ya han sido declaradas.

Para que quede claro este asunto de la colocación de funciones veamos el siguiente ejemplo, **que daría un error**. Examina atentamente el código siguiente, que lanzará un error, debido a que hacemos 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>
```

Con esto esperamos haber resuelto todas las dudas sobre la colocación del código de las funciones Javascript. En siguientes artículos veremos otros temas interesantes como los parámetros de las funciones.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *02/11/2001*
Disponible online en *http://desarrolloweb.com/articulos/20.php*

Parámetros de las funciones

Vemos lo que son los parámetros en las funciones. Vemos como definir funciones que reciben parámetros en el lenguaje Javascript y como hacer llamadas a funciones pasando parámetros.

En el Manual de Javascript hemos hablado anteriormente sobre funciones. En concreto este es el tercer artículo que abordamos sobre el tema.

Las ideas que hemos explicado 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 de datos. En este artículo veremos cómo podemos enviar datos a las funciones Javascript.

Parámetros

Los parámetros se usan para mandar valores a las funciones. Una función trabajará con los parámetros para realizar las acciones. Por decirlo de otra manera, los parámetros son los valores de entrada que recibe una función.

Por poner un ejemplo sencillo de entender, una función que realizase una suma de dos números tendría como parámetros a esos dos números. Los dos números son la entrada, así como la salida sería el resultado de la suma, 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 donde recibimos el parámetro 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, para asegurarnos que todo es consecuente con los tipos de datos que esperamos tengan nuestras variables o parámetros.

Múltiples parámetros

Una función puede recibir tantos parámetros como queramos y para expresarlo se colocan los nombres de los parámetros separados por comas, dentro de los paréntesis. Veamos rápidamente la sintaxis para que la función de antes, pero hecha para que 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.

Los parámetros se pasan por valor

Al hilo del uso de parámetros en nuestros programas Javascript, tenemos que saber que los parámetros de las funciones se pasan por valor. Esto quiere decir que estamos pasando valores y no variables. En la práctica, 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, ésta no cambia de valor.

De este modo, una vez ejecutada la función, 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 con el que habíamos actualizado el parámetro.

En Javascript sólo se pueden pasar las variables por valor.

Ahora que hemos aprendido a enviar datos a las funciones, por medio de los parámetros, podemos

aprender a hacer funciones que devuelven valores.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *02/11/2001*
Disponible online en *http://desarrolloweb.com/articulos/20.php*

Valores de retorno

Las funciones pueden devolver valores, a través de la sentencia return. También vemos un apunte sobre el ámbito de variables en funciones en Javascript.

Estamos aprendiendo acerca del uso de funciones en Javascript y en estos momentos quizás ya nos hayamos dado cuenta de la gran importancia que tienen para hacer programas más o menos avanzados. En este artículo del Manual de Javascript seguiremos aprendiendo cosas sobre funciones y en concreto que con ellas también se puede devolver valores. Además, veremos algún caso de uso interesante sobre las funciones que nos puede aclarar un poco el ámbito de variables locales y globales.

Devolución de valores en las funciones

Las funciones en Javascript también pueden retornar valores. De hecho, ésta es una de las utilidades más esenciales de las funciones, que debemos conocer, no sólo en Javascript sino en general en cualquier lenguaje de programación. De modo que, al invocar una función, se podrá realizar acciones y ofrecer un valor como salida.

Por ejemplo, una función que calcula el cuadrado de un número tendrá como entrada a ese número y como salida tendrá el valor resultante de hallar el cuadrado de ese número. La entrada de datos en las funciones la vimos anteriormente en el artículo sobre parámetros de las funciones. Ahora tenemos que aprender acerca de la salida.

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 calculada de los dos números.

Quizás nos preguntemos ahora cómo recibir un dato que devuelve una función. Realmente en el código fuente de nuestros programas podemos invocar a las funciones en el lugar que deseemos. Cuando una función devuelve un valor simplemente se sustituye la llamada a la función por ese valor que devuelve. Así

pues, para almacenar un valor de devolución de una función, tenemos que asignar la llamada a esa función como contenido en una variable, y eso lo haríamos con 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 realidad en Javascript las funciones sólo pueden devolver un valor, por lo que en principio no podemos hacer funciones que devuelvan dos datos distintos.

> **Nota:** en la práctica nada nos impide que una función devuelva más de un valor, pero como sólo podemos devolver una cosa, tendríamos que meter todos los valores que queremos devolver en una estructura de datos, como por ejemplo un array. No obstante, eso sería un uso más o menos avanzado que no vamos a ver en estos momentos.

Ahora bien, aunque sólo podamos devolver un dato, en una misma función podemos colocar más de un return. Como decimos, 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. Sobre este asunto debemos de saber que todas las variables declaradas en una función son locales a esa función, es decir, sólo tendrán validez durante la ejecución de la función.

Nota: Incluso, si lo pensamos, nos podremos dar cuenta que los parámetros son como variables que se declaran en la cabecera de la función y que se inicializan al llamar a la función. Los parámetros también son locales a la función y tendrán validez sólo cuando ésta se está ejecutando.

Podría darse el caso de que 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.

Veamos el siguiente código.

```
function variables_glogales_y_locales(){
   var variableLocal = 23
   variableGlobal = "qwerty"
}
```

En este caso variableLocal es una variable que se ha declarado en la función, por lo que será local a la función y sólo tendrá validez durante su ejecución. Por otra parte variableGlobal no se ha llegado a declarar (porque antes de usarla no se ha utilizado la palabra var para declararla). En este caso la variable variableGlobal es global a toda la página y seguirá existiendo aunque la función finalice su ejecución. Además, si antes de llamar a la función existiese la variable variableGlobal, como resultado de la ejecución de esta función, se machacaría un hipotético valor de esa variable y se sustituiría por "qwerty".

Nota: Podemos encontrar más información sobre ámbito de variables en un artículo anterior.

Con esto hemos terminado el tema de las funciones, así que en adelante nos dedicaremos a otros asuntos también interesantes, como son los Arrays en Javascript.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *02/11/2001*
Disponible online en *http://desarrolloweb.com/articulos/20.php*

Libreria de funciones Javascript

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

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

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

Funciones incorporadas en Javascript

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

eval(string)

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

parseInt(cadena,base)

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

parseFloat(cadena)

Convierte la cadena en un número y lo devuelve.

escape(carácter)

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

unescape(carácter)

Hace exatamente lo opuesto a la función escape.

isNaN(número)

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

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

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

Ejemplos de uso de las funciones incorporadas en Javascript

Hasta el momento hemos conocido simplemente un listado de las funciones nativas del lenguaje Javascript. Ahora podemos ver varios ejemplos de utilización de funciones nativas de Javascript, que tenemos disponibles en cualquier navegador y en cualquier versión de Javascript.

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

Función eval

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

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

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

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

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

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

Función parseInt

Esta función recibe un número, escrito como una cadena de caracteres, y un número que indica una base. En realidad puede recibir otros tipos de variables, dado que las variables no tienen tipo en Javascript, pero

se suele utilizar pasándole un string para convertir la variable de texto en un número.

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

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

```
document.write (parseInt("34"))
```

Devuelve el numero 34

```
document.write (parseInt("101011",2))
```

Devuelve el numero 43

```
document.write (parseInt("34",8))
```

Devuelve el numero 28

```
document.write (parseInt("3F",16))
```

Devuelve el numero 63

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

```
document.write (parseInt("3.38"))
```

Devuelve el numero 3

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

```
document.write (parseInt("desarrolloweb.com"))
```

Devuelve el numero NaN

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

```
document.write (parseInt("16XX3U"))
```

Devuelve el numero 16

```
document.write (parseInt("TG45"))
```

Devuelve el numero NaN

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

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

Función isNaN

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

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

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

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

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

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

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

Esperamos que los ejemplos vistos en este artículo hayan resultado interesantes. No obstante, como

habíamos señalado anteriormente, existen bastantes otras funciones nativas en Javascript que debemos conocer, pero que están asociadas a clases y objetos nativos Javascript. Pero antes de pasar a ese punto queremos ofrecer una pequeña guía básica para el trabajo con programación orientada a objetos en Javascript.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *11/03/2002*
Disponible online en *http://desarrolloweb.com/articulos/20.php*

Arrays Javascript

Los arrays, también llamados tablas o matrices, son la primera estructura de datos que podemos aprender en Javascript y en otros lenguajes de programación. Sin duda serán imprescindibles para desarrollar programas medianamente avanzados.

Arrays en Javascript

Vemos que son los arrays en Javascript, para qué sirven y cómo utilizarlos. Veremos diversas formas de crearlos, así como definir y acceder a sus valores.

Pasamos a un nuevo tema en el Manual de Javascript, en el que vamos a conocer nuestra primera estructura de datos.

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 la 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 compartimiento o posición a la que nos estamos refiriendo.

> **Nota:** 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 la verdad es que actualmente esta limitación no debe preocuparnos. Además, dada la complejidad de la tarea de simular un array por medio de objetos, por lo menos en el momento en que nos encontramos y las pocas ocasiones en que lo necesitaremos, opinamos que es mejor olvidarnos de ese asunto y trabajar simplemente con los arrays normalmente. Así que en este artículo y los siguientes 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 en Javascript empiezan siempre en la posición 0, así que un array que tenga por ejemplo 10 posiciones, tendrá casillas de la 0 a la 9. Para recoger datos de un array lo hacemos igual: poniendo entre corchetes el índice de la posición a la que queremos acceder. Veamos cómo se imprimiría en la pantalla el contenido de un array.

```
var miArray = new Array(3)

miArray[0] = 155
miArray[1] = 4
miArray[2] = 499

for (i=0;i<3;i++){
	document.write("Posición " + i + " del array: " + miArray[i])
	document.write("<br>")
}
```

Hemos creado un array con tres posiciones, luego hemos introducido un valor en cada una de las posiciones del array y finalmente las hemos impreso. En general, el recorrido por arrays para imprimir sus posiciones, o cualquier otra cosa, se hace utilizando bucles. En este caso utilizamos un bucle FOR que va desde el 0 hasta el 2.

Podemos ver el ejemplo en marcha en otra página.

Tipos de datos en los arrays

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

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

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

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

Declaración e inicialización resumida de Arrays

En Javascript tenemos a nuestra disposición una manera resumida de declarar un array y cargar valores en un mismo paso. Fijémonos en el código siguiente:

```
var arrayRapido = [12,45,"array inicializado en su declaración"]
```

Como se puede ver, se está definiendo una variable llamada arrayRapido y estamos indicando en los corchetes varios valores separados por comas. Esto es lo mismo que haver declarado el array con la función Array() y luego haberle cargado los valores uno a uno.

En el próximo artículo seguiremos viendo cosas relacionadas con los arrays, en concreto aprenderemos a acceder a la longitud de un array.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *22/12/2001*
Disponible online en *http://desarrolloweb.com/articulos/20.php*

Longitud de los arrays

Aprendemos más cosas sobre el funcionamiento de los arrays y en concreto vemos como utilizar su propiedad length para acceder al número de casillas que tiene.

En el artículo anterior del Manual de Javascript empezamos a explicar el concepto de array y su utilización en Javascript. En este artículo vamos a continuar con el tema, mostrando el uso de su propiedad length.

Todos los arrays en javascript, aparte de almacenar el valor de cada una de sus casillas, 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 que tiene el 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.

Continuaremos el tema de arrays en la siguiente entrega de este manual: Arrays multidimensionales.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *22/12/2001*
Disponible online en *http://desarrolloweb.com/articulos/20.php*

Arrays multidimensionales en Javascript

Vemos qué son los arrays multidimensionales (arrays de más de una dimensión) y cómo utilizarlos. Además explicamos cómo inicializar arrays en su declaración.

Como estamos viendo, los arrays son bastante importantes en Javascript y también en la mayoría de los lenguajes de programación. En concreto ya hemos aprendido a crear arrays y utilizarlos en artículos anteriores del Manual de Javascript. Pero aun nos quedan algunas cosas importantes que explicar, como son los arrays de varias dimensiones.

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-multidimensinal. 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 bidimensinal.

También es interesante ver cómo se realiza un recorrido por un array de dos dimensiones. Para ello tenemos que hacer un bucle que pase por cada una de las casillas del array bidimensional y dentro de éstas 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. 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 así 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.

> **Nota:** Habrás podido observar que en ocasiones generar código HTML desde Javascript se hace complejo. Pero el problema no es solo que el código sea difícil de producir, sino lo peor es que creas un código difícil de mantener, en el que se mezcla tanto la parte de la programación en Javascript con la parte de la presentación en HTML. Lo que has visto además es solo un código bien simple, con una tabla realmente elemental, imagina qué pasaría cuando la tabla o los datos fueran más complejos. Afortunadamente, hay maneras de generar código HTML de salida mejores que las que hemos visto ahora, aunque resulta un poco avanzado para el momento en el que estamos. De todos modos, te dejamos un enlace al manual del sistema de templates Javascript Handlebars, que es una alternativa de librería sencilla para generar salida en HTML desde Javascript.

Podemos ver el ejemplo en marcha y examinar el código del script entero.

Inicialización de arrays

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

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

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

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

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

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

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

Javascript todavía tiene una manera más resumida que la que acabamos de ver, que explicamos en el primer artículo donde tratamos los arrays. Para ello simplemente escribimos entre corchetes los datos del array que estamos creando. Para acabar vamos a mostrar un ejemplo sobre cómo utilizar esta sintaxis para declarar arrays de más de una dimensión.

```
var arrayMuchasDimensiones = [1, ["hola", "que", "tal", ["estas", "estamos", "estoy"], ["bien", "mal"], "acabo"], 2, 5];
```

En este ejemplo hemos creado un array muy poco uniforme, porque tiene casillas con contenido de simples enteros y otras con contenido de cadena y otras que son otros arrays. Podríamos acceder a algunas de sus casillas y mostrar sus valores de esta manera:

```
alert (arrayMuchasDimensiones[0])
alert (arrayMuchasDimensiones[1][2])
alert (arrayMuchasDimensiones[1][3][1])
```

Con esto hemos llegado al fin de los artículos que tratan sobre arrays en Javascript y ahora podemos continuar con una pequeña pausa y consejos que vendrán bien para mejorar nuestra relación con este lenguaje de programación.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *22/12/2001*
Disponible online en *http://desarrolloweb.com/articulos/20.php*

Objetos en Javascript

En los siguientes artículos vamos a abordar el mundo de los objetos en Javascript. Serán esenciales para cualquier persona que está comenzando en la programación en general, ya que se tratarán conceptos muy recurrentes en cualquier lenguaje. Además trataremos con detalle las particularidades de los objetos en el lenguaje Javascript, ya que son bastante distintas en relación a otros lenguajes más tradicionales.

Introducción general a los objetos en Javascript

Breve introducción al mundo de los objetos, en programación en general, y a las particularidades del uso de objetos en el lenguaje Javascript.

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

La programación orientada a objetos (POO) representa una nueva manera de pensar a la hora de hacer un programa. Javascript no es un lenguaje de programación orientado a objetos puro porque, aunque utiliza objetos en muchas ocasiones, no necesitamos programar todos nuestros programas en base a ellos. De hecho, lo que vamos a hacer generalmente con Javascript es **usar objetos** y no tanto programar orientado a objetos. Por ello, la manera de programar no va a cambiar mucho con respecto a lo que hemos visto hasta ahora en el Manual de Javascript. En resumen, lo que hemos visto hasta aquí relativo a sintaxis, funciones, etc. sigue siendo perfectamente válido y puede ser utilizado igual que se ha indicado. Solo vamos a aprender una especie de estructura nueva como son los objetos.

> **Nota:** Para empezar a empaparnos un poco sobre los objetos tenemos un pequeño artículo publicado en DesarrolloWeb sobre la programación orientada a objetos. Sería muy recomendable que lo leyeras, porque se explican varios conceptos en los cuales no vamos a entrar con tanto detalle. Si conoces ya la POO continúa leyendo sin pausa, pero si deseas profundizar recuerda que tenemos también un Manual completo de Orientación a objetos. Si te gusta ver vídeos te recomendamos también la clase Qué son los objetos impartida en el Curso de Programación en vídeo.

Qué es un objeto

Aunque no vamos a entrar en detalle con los concetos, pues se encuentran muy bien explicados en referencias que ya hemos indicado, los objetos son una herramienta de lenguajes de programación en la que se unen dos cosas fundamentales: los datos y la funcionalidad. Todo programa informático trata básicamente esas dos cosas de alguna manera. Con lo que hemos visto hasta ahora los datos los teníamos en

variables y la funcionalidad en funciones ¿no es así? pues en el mundo de los objetos, tanto datos como funcionalidad están en la misma estructura, el objeto.

El asunto es que ahora necesitas aprender nuevos nombres con los que referirte a los datos y funcionalidad agrupados en un objeto:

- **Propiedades**: En los objetos las propiedades se refieren a los datos
- **Métodos**: En objetos, los métodos se refieren a la funcionalidad

Imagina que tienes un objeto botón (un botón del navegador, algo que puedes pulsar para realizar una acción). El botón tiene un texto escrito, pues ese texto sería un dato y por lo tanto le llamaríamos propiedad. Otra propiedad de un botón sería si está o no activado. Por otra parte, un botón podría tener funcionalidad asociada, que estaría en un método, como procesar la acción de un clic. Imagina algo más genérico como un teléfono. El teléfono puede tener propiedades como la marca, modelo, sistema operativo y métodos como encender, apagar, llamar a un número, etc.

En lenguajes de programación orientados a objetos puros, como puede ser Java, tienes que programar siempre en base a objetos. Para programar tendrías que crear "clases", que son una especie de "moldes" a partir de los cuales se crean objetos. El programa resolvería cualquier necesidad mediante la creación de objetos en base a esos moldes (clases), existiendo varios (decenas, cientos o miles) de objetos de diversas clases. Los objetos tendrían que colaborar entre si para resolver cualquier tipo de acción, igual que en sistemas como un avión existen diversos objetos (el motor, hélices, mandos...) que colaboran entre sí para resolver la necesidad de llevar pasajeros o mercancía en viajes aéreos.

Sin embargo, como veníamos diciendo, en Javascript no es tanto programar orientado a objetos, sino usar objetos. Muchas veces serán objetos ya creados por el propio navegador (la ventana del navegador, un documento HTML que se está visualizando, una imagen o un formulario dentro de ese documento HTML, etc), y otras veces serán objetos creados por ti mismo o por otros desarrolladores que te sirven para hacer cosas específicas. Por tanto, lo que nos interesa saber para comenzar es la sintaxis que necesitas para usar los objetos, básicamente acceder a sus propiedades y ejecutar sus métodos.

> **Nota:** Para conocer cuáles son los objetos del navegador, que tenemos a disposición en Javascript para resolver las necesidades de las páginas web, tienes que leer el manual sobre trabajo con Javascript para uso y manipulación de los recursos del navegador.

Cómo acceder a propiedades y métodos de los objetos

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

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

```
miObjeto.miPropiedad
```

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

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

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

```
miObjeto.miMetodo()
```

Cómo crear objetos

Como hemos dicho, la mayoría de los objetos con los que vas a trabajar en Javascript para poder crear interacción, efectos y comportamientos diversos en páginas web, te los dan ya hechos. El propio navegador te los ofrece para que tú simplemente los tengas que usar. Eso es material de estudio del Manual de Javascript y los objetos del navegador. Aclarado ese punto hay que advertir que Javascript es un tanto particular a la hora de crear objetos, básicamente porque tradicionalmente no existe el conceto de "clase".

Para ser más exactos, en Javascript, ES5, las clases se crean por medio de funciones y con el operador new creas objetos a partir de esas funciones, pero no existen las clases como las que conocemos en otros lenguajes más tradicionales.

> **Nota:** Ahora en ES6 ya existen las clases y Javascript es capaz de generar clases y a partir de ellas producir objetos, como otros lenguajes. Obtienes más información en el Manual de ES6.

La otra alternativa para crear objetos en Javascript es por medio de literales de objeto, que no son más que la definición del objeto por medio de código encerrado entre llaves, indicando sus propiedades o métodos tal cual.

```
{
   nombre: 'Miguel Angel Alvarez',
   sitioWeb: 'DesarrolloWeb.com'
}
```

En el código anterior solo hemos definido propiedades, pero tenemos otros artículos donde podrás ver cómo definir métodos también. Para saber más sobre los literales de objetos te recomendamos la lectura del artículo sobre Literales de objetos Javascript, donde vamos a explicarte más detenidamente esta sintaxis habitual de creación de objetos en este lenguaje.

En Javascript tradicional hemos dicho que no existen las clases, pero podremos crear instancias de objetos a partir de funciones, como veremos en el siguiente punto.

Crear e instanciar objetos a partir de funciones

Para quien no lo sepa, instanciar un objeto es la acción de crear un ejemplar de una clase, para poder trabajar con él luego. La clase es la definición de las características y funcionalidades de un objeto. Con las clases no se trabaja directamente, éstas sólo son definiciones. Para trabajar con una clase debemos tener un objeto instanciado de esa clase. Recordamos que en Javascript no existen clases, pero podemos usar funciones.

Esta simple función podríamos usarla como molde para construir objetos de la clase Persona:

```
function Persona(nombre) {
  this.nombre = nombre;
}
```

Observarás que estamos usando dentro la palabra "this". Esa palabra es una referencia al objeto que se va a crear con esta función. En javascript para crear un objeto a partir de una función se utiliza la instrucción new, de esta manera.

```
var miguel = new Persona('Miguel Angel Alvarez');
```

En una variable que llamamos "miguel" asigno un nuevo (new) ejemplar de la clase Persona. Los paréntesis se rellenan con los datos que necesite la clase para inicializar el objeto, si no hay que meter ningún parámetro los paréntesis se colocan vacíos. En realidad lo que se hace cuando se crea un objeto es llamar a la función que lo construye y la propia función se encargará de inicializar las propiedades del objeto (para lo que usa la referencia "this").

Ciertamente, si los conceptos de programación orientada a objetos son nuevos para ti, quizás muchos puntos de este artículo se quedarán un poco complejos. No queremos que te asustes, puesto que volveremos sobre todo esto en futuros artículos y lo podrás ir entendiendo mejor. En concreto, esta parte de la creación de objetos a partir de funciones está explicada en el artículo Creación de clases en Javascript con ES5. Recuerda también aclarar tus dudas más teóricas sobre clases y objetos en el Manual de la teoria de la programación orientada a objetos.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *19/12/2016*
Disponible online en *http://desarrolloweb.com/articulos/introduccion-objetos-javascript.html*

Literales de objeto en Javascript

Cómo podemos crear objetos en Javascript a partir de un literal, operaciones típicas que puedes realizar con esos objetos.

Voy a comenzar una serie de artículos que me sirven como apuntes de programación orientada a objetos en Javascript. Este lenguaje es bastante particular en este sentido y en ocasiones te permite hacer cosas que en otros lenguajes serían impensables. También al contrario, algunas cosas básicas de lenguajes tradicionales en Programación Orientada a Objetos no existen en Javascript.

Javascript no tiene clases como tal, por lo menos hasta que llegue la nueva actualización Ecma6 prevista para 2015, pero sí que tiene objetos. Además tiene algo que no todos los lenguajes poseen, un mecanismo para la creación de objetos a partir de lo que conocemos como "literal".

Nota: Para los que no conozcan el término "literal", cabe decir que es una palabra que indica algo escrito de manera "literal". Aunque sea muy feo usar la misma palabra para definir algo, me lo vas a permitir porque es la mejor que existe. Un literal es algo extremadamente sencillo de entender con un ejemplo.

```
var x = "hola";
```

En esa línea de código "hola" es un literal. En concreto decimos que es un "literal de cadena". Ahora mira esta otra línea:

```
y +=5;
```

Cuando ves un número escrito tal cual en tu código decimos que es un literal numérico.

Podemos crear un objeto en Javascript asignando un valor literal de objeto en una variable. Eso se consigue colocando dicho literal entre llaves y dentro de ellas tantas propiedades o métodos con pares "clave/valor", por medio de una sintaxis como esta:

```
var dimensiones = {
    altura: 34,
    anchura: 455
}
```

Como estás viendo, tenemos una variable dimensiones. Al asignarle un literal objeto estamos realmente asignando una referencia a un objeto en la memoria creado con las propiedades que acabamos de asignar.

Las propiedades se separan por comas y se coloca siempre el nombre de la propiedad, el caracter ":" y luego el valor de la propiedad.

Por supuesto, también podremos asignar métodos a nuestros objetos literales.

```
var dimensiones = {
    altura: 34,
    anchura: 455,
    area: function(){
      return this.altura * this.anchura;
    }
}
```

Como ves, los métodos en Javascript simplemente son propiedades a los que les asignas una función. Dentro del código de tus métodos puedes acceder a las propiedades del objeto a través de la variable this.

Una vez creado ese objeto, puedes usar la notación punto para acceder a sus propiedades (o métodos).

```
dimensiones.altura = 90 //accede a la propiedad altura y le asigna el valor 90
```

Nota: Los literales de objeto no son la única manera de crear objetos en Javascript. Además existe un manera de definir algo parecido a una clase, pero no es exactamente lo que conocemos en la programación orientada a objetos tradicional y no podríamos llegar a considerarla tal. En lugar de ello podemos crear funciones que, al invocarlas con la palabra "new" te crean nuevos objetos inicializados con esa función. Técnicamente, en vez de definir clases, en Javascript definimos funciones constructoras de objetos. En definitiva, una implementación muy particular que a veces nos puede liar si estamos familiarizados con lenguajes de enfoque más tradicional (en lo que respecta a OOP "Object Oriented Programing") como Java o PHP. Esta es una discusión interesante, que podrías complementar con el artículo dedicado a crear clases en Javascript, pero que no es la que nos ocupa en esta ocasión.

¿Esto no es JSON?

El formato de intercambio de datos JSON se ha popularizado mucho y quizás lo conozcas antes de conocer estos detalles sobre los literales de objeto Javascript. En ese caso puede que te hayas dado cuenta que los literales de objeto Javascript no son más que estructuras JSON. Si es así permíteme apuntar que realmente tendríamos que darle la vuelta a esa frase y decir que la notación JSON utiliza la sintaxis de los literales de objeto Javascript como formato.

Así pues, lo que conozcas de JSON lo puedes aplicar al mundo de Javascript inmediatamente. De hecho, si has tenido ocasión de trabajar con este formato desde Javascript, habrás comprobado que en la mayoría de los casos puedes volcar un objeto JSON en una variable Javascript e inmediatamente trabajar con él como si fuera un objeto que tienes en la memoria.

Nota: Puedes saber más de JSON en el artículo Qué es JSON.

Crear nuevas propiedades y métodos sobre objetos creados

Javascript es muy permisivo y nos deja hacer cosas que producirían errores en otros lenguajes. Es el caso de

la asignación de valores a propiedades que no han sido creadas previamente.

Tengo mi objeto coche:

```
var coche = {
       color: "rojo",
       marca: "Opel"
}
```

Ahora podría crear nuevas propiedades en ese objeto asignando valores a las propiedades que no existían previamente.

```
coche.anoFabricacion = 2014;
```

Los métodos los creas asignando funciones:

```
coche.arrancar = function(){
      alert("rum rum");
}
```

Los métodos y funciones que acabamos de crear son tan válidos como los que ya estuvieran en mi objeto cuando fue definido por medio de su literal. Podré acceder a sus elementos con la notación punto, que ya conoces.

```
console.log(coche.color);
console.log(coche.anoFabricacion);
coche.arrancar();
```

Ejemplo para practicar con literales de objeto

En el siguiente código ponemos en marcha los nuevos conocimientos que has adquirido sobre literales de objeto en Javascript.

```
var miObjeto = {
    propiedad1: "Algo",
    propiedad2: "Otra cosa",
    propiedad3: true,
    propiedad4: 344,

    metodo1: function(){
          alert("Ejecutaste metodo1");
    },
    metodo2: funcionMetodo2
}
```

```
function funcionMetodo2(){
    //puedo usar la variable this para acceder a mis propias propiedades o métodos
    this.propiedad2 = "Esto lo he modificado desde el método metodo2";
}

//Veamos el valor de esas propiedades
console.log(miObjeto.propiedad1);
console.log(miObjeto.propiedad2);

//ejecutemos algún método
miObjeto.metodo1();
miObjeto.metodo2();

//creamos nuevas propiedades
miObjeto.otraPropiedad = "Esto está creado a posteriori";

//creamos nuevos métodos
miObjeto.otroMetodo = function(){
    console.log("ejecutaste otro método");
}

//veamos el contenido de todo el objeto
console.log(miObjeto);
```

Eso es todo por el momento, espero que te hayamos aclarado algo.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *31/10/2014*
Disponible online en *http://desarrolloweb.com/articulos/literales-objeto-javascript.html*

for in en Javascript

Recorridos for in a propiedades de objetos en Javascript. Cómo iterar por las propiedades y los valores de las propiedades de un objeto, de manera genérica en Javascript con el bucle for ... in.

En Javascript no existen arrays asociativos y como sabes, éstos son siempre buenos aliados como recursos para la programación. Si queremos usar algo parecido a un array asociativo tendremos que utilizar las construcciones de objetos. De todos modos, con lo que nos ofrece el lenguaje somos capaces de realizar todas las cosas que en otros lenguajes haces con los arrays asociativos.

En este artículo pretendemos explicarte cómo realizar un recorrido genérico a todas las propiedades presentes en un objeto, junto con sus valores. Ese recorrido es "genérico", es decir, es independiente del número de propiedades que se encuentre en el objeto que estamos recorriendo y es independiente también de sus nombres de propiedad o sus valores.

Nota: Por si no lo sabes, los arrays asociativos son aquellos que no tienen índices numéricos sino alfabéticos. Nos sirven para muchas cosas en los lenguajes de programación. Si no lo sabías, ya te darás

cuenta. De momento entonces olvida la mención a los arrays asociativos y piensa solo en objetos y sus propiedades.

Bucle for ... in

En Javascript hay una construcción especial del bucle for de toda la vida, que te permite recorrer todas las propiedades de un objeto. Es parecido a lo que en otros lenguajes tenemos en el bucle foreach (Javascript también tiene el forEach pero es solo para arrays y no es un bucle sino un método de arrays, que sirve para iterar, pero no es una estructura de control como tal). Su sintaxis es la siguiente:

```
for (propiedad in objeto){
    //código a repetir por el bucle.
    //dentro de este código la variable "propiedad" contiene la propiedad actual
//actual en cada uno de los pasos de la iteración.
}
```

No hay mucho más que explicar sobre esta estructura del bucle for-in, solo decir que de esta manera tal cual puedes acceder a los nombres de las propiedades del objeto. Gracias al bucle, el código que se incluye dentro del for se ejecutará una vez por cada una de las propiedades del objeto.

Habrás notado que hemos dicho que así podrás acceder a las propiedades. No te preocupes que para acceder a sus valores tendrás que usar un truquillo. Lo veremos también en este artículo, pero no nos adelantemos.

Bucle para acceder a las propiedades de un objeto

Veamos una aplicación de este bucle for in con un ejemplo sencillo que nos arroje algo de luz.

Tenemos un objeto creado a partir de un literal de objeto.

```
var diasMes = {
    enero: 31,
    febrero: 28,
    marzo: 31,
    abril: 30,
    mayo: 31
}
```

Son los meses del año y los días que tiene ese mes. Es verdad que febrero puede tener otros días y que me faltan meses, pero a fines didácticos es suficiente. Observarás que esta estructura en resumen es muy parecida a lo que sería un array asociativo. Si los necesitamos en nuestro programa la podemos usar como si lo fuera.

> **Nota:** Si no entiendes la definición de un objeto por medio de un literal, simplemente lee el artículo sobre literales de objeto.

Pues para acceder a las propiedades de ese objeto (en este caso sería para acceder a los nombres de cada uno de los meses) usarías este bucle.

```
for (var mes in diasMes){
    console.log(mes);
}
```

Con esto conseguirás en la consola que aparezcan todos los nombres de los meses del año que tenemos en ese objeto diasMes.

Bucle para acceder a los valores de las propiedades

Una situación común es que no quieras acceder a los nombres de las propiedades, sino a los valores. Para ello podemos usar un pequeño truco que nos permite Javascript. Se trata de que podemos acceder a las propiedades de los objetos como si fueran arrays, indicando como índice del array una cadena con el nombre de la propiedad.

```
alert(diasMes["mayo"]);
```

Eso nos mostrará en una caja de alerta el valor 31 que es el que está asociado a la propiedad "mayo" del objeto "diasMes".

Te habrás dado cuenta que esta sintaxis para acceder a los valores de las propiedades es justamente igual a la sintaxis que se usa para acceder a valores de casillas de un array asociativo.

Bueno, pues sabiendo esto podrás acceder a los valores del objeto, uno por uno, con un bucle for ... in muy parecido al que teníamos antes.

```
for (var mes in diasMes){
    console.log(diasMes[mes]);
}
```

Seguro que ahora te parece sencillo. Es que realmente lo es.

Todo junto, acceso a la propiedad y su valor

Esta parte ya seguro que ni te hace falta leerla. Simplemente quiero mostrar un mensaje en la consola más legible, accediendo en el mismo bucle a la propiedad y su valor.

```
for (var mes in diasMes){
    console.log("El mes " + mes + " tiene " + diasMes[mes] + " dias.");
}
```

Eso es todo, con estos conocimientos ya no se te pueden escapar las posibilidades de recorridos a objetos y la construcción del útil bucle for-in. Además, si necesitas arrays asociativos para construir tus programas sabrás la manera alternativa de implementarlos y usarlos, por medio de objetos, en Javascript. Seguro que te servirá de ayuda, incluso aunque ahora no sepas para qué.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *19/11/2014*
Disponible online en *http://desarrolloweb.com/articulos/recorridos-propiedades-objetos-javascript-forin.html*

Creación de clases en Javascript tradicional

Ahora que ya sabemos lo que son los objetos, vamos a ver cómo podemos crear y usar nuestros propios objetos en Javascript tradicional, ES5, abordando diversos ejemplos prácticos.

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

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

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

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

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

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

La clase AlumnoUniversitario

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

Constructor: Colocamos propiedades

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

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

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

Escribir métodos

Ahora vamos a continuar con esa definición de la clase para incorporar métodos.

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

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

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

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

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

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

Constructor: Colocamos métodos

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

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

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

Para instanciar un objeto

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

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

Trabajando con la clase, creando instancias y usando objetos

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

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

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

```
        document.write("<br>Número de matrícula: " + this.numMatricula)
}

//definimos el constructor para la clase
function AlumnoUniversitario(nombre, edad){
        this.nombre = nombre
        this.edad = edad
        this.numMatricula = null
        this.matriculate = matriculate
        this.imprimete = imprimete
}

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

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

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

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

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

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

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *21/05/2002*
Disponible online en *http://desarrolloweb.com/articulos/clases-javascript-es5.html*

Librerías de clases y objetos en Javascript

En Javascript, como en cualquier lenguaje, existen muchas funcionalidades básicas ya desarrolladas para el resolver necesidades habituales en los programas, como cálculos matemáticos, o control de fechas. En los siguientes artículos podrás conocer cómo esas librerías están disponibles en Javascript por medio de diversas clases y objetos de utilidad general. Aprenderemos y practicaremos con diversos recursos del lenguaje para realizar todo tipo de operaciones matemáticas avanzadas, trabajo con cadenas de caracteres, trabajo con fechas, tipos primitivos, etc.

Objetos incorporados en Javascript

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

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

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

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

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

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

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

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

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

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

Ejemplos:

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

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

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *21/03/2002*
Disponible online en *http://desarrolloweb.com/articulos/20.php*

Tratamiento de String en Javascript

Las cadenas en Javascript son objetos de la clase String, que contienen diversos métodos que nos sirven para manejar cadenas de caracteres. Estudiamos sus propiedades y la lista completa de métodos.

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

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

Propiedades de String

Length

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

Métodos de String

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

charAt(indice)

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

indexOf(carácter,desde)

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

lastIndexOf(carácter,desde)

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

replace(substring_a_buscar,nuevoStr)

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

split(separador)

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

substring(inicio,fin)

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

toLowerCase()

Pone todas los caracteres de un string en minúsculas.

toUpperCase()

Pone todas los caracteres de un string en mayúsculas.

toString()

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

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

anchor(name)

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

big()

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

blink()

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

bold()

Como si utilizásemos la etiqueta `<B>`.

fixed()

Para utilizar una fuente monoespaciada, etiqueta `<TT>`.

fontColor(color)

Pone la fuente a ese color. Como utilizar la etiqueta `<FONT color=el_color_indicado>`.

fontSize(tamaño)

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

italics()

Pone la fuente en cursiva. Etiqueta <I>.

link(url)

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

small()

Es como utilizar la etiqueta <SMALL>

strike()

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

sub()

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

sup()

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

Ejemplos de funcionamiento de los objetos de clase String

Hasta ahora hemos visto muchos métodos interesantes de elementos de tipo String, pero no estará de más pasar a la práctica para entender todo mucho mejor. Así pues, veamos unos ejemplos sobre cómo se utilizan los métodos y propiedades del objeto String.

Ejemplo de strings 1

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

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

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

document.write(result)
```

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

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

Ejemplo de strings 2

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

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

posicion_mitad = miString.length / 2

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

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

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

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

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

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

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *21/03/2002*
Disponible online en *http://desarrolloweb.com/articulos/objetos-string-javascript.html*

Clase Date en Javascript

Explicamos la clase Date de Javascript, que se utiliza para el manejo de fechas y horas.

Exploramos sus diversos métodos y propiedades de los objetos Date, con los que realizar cálculos en el tiempo.

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

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

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

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

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

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

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

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

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

getDate()

Devuelve el día del mes.

getDay()

Devuelve el día de la semana.

getHours()

Retorna la hora.

getMinutes()

Devuelve los minutos.

getMonth()

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

getSeconds()

Devuelve los segundos.

getTime()

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

getYear()

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

getFullYear()

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

setDate()

Actualiza el día del mes.

setHours()

Actualiza la hora.

setMinutes()

Cambia los minutos.

setMonth()

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

setSeconds()

Cambia los segundos.

setTime()

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

setYear()

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

setFullYear()

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

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

Ejemplos de trabajo con fechas en Javascript

Visto lo anterior, ahora podemos poner los conocimientos en la práctica en un ejemplo completo de trabajo con objetos de la clase Date.

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

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

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

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

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

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

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

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

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

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

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

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

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

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *15/04/2002*
Disponible online en *http://desarrolloweb.com/articulos/clase-date-javascript.html*

Clase Math en Javascript

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

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

De modo que para cualquier cálculo matemático complejo utilizaremos la clase Math, con una particularidad. Hasta ahora cada vez que queríamos hacer algo con una clase debíamos instanciar un objeto de esa clase y trabajar con el objeto y en el caso de la clase Math se trabaja directamente con la clase. Esto se permite por que las propiedades y métodos de la clase Math son lo que se llama propiedades y métodos de

clase y para utilizarlos se opera a través de la clase en lugar de los objetos. Dicho de otra forma, para trabajar con la clase Math no deberemos utilizar la instrucción new y utilizaremos el nombre de la clase para acceder a sus propiedades y métodos.

Propiedades de Math

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

E

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

LN2

Logaritmo neperiano de 2.

LN10

Logaritmo neperiano de 10.

LOG2E

Logaritmo en base 2 de E.

LOG10E

Logaritmo en base 10 de E.

PI

Conocido número para cálculo con círculos.

SQRT1_2

Raiz cuadrada de un medio.

SQRT2

Raiz cuadrada de 2.

Métodos de Math

Así mismo, tenemos una serie de métodos para realizar operaciones mateméticas típicas, aunque un poco

complejas. Todos los que conozcan las matemáticas a un buen nivel conocerán el significado de estas operaciones.

abs()

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

acos()

Devuelve el arcocoseno de un número en radianes.

asin()

Devuelve el arcoseno de un numero en radianes.

atan()

Devuelve un arcotangente de un numero.

ceil()

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

cos()

Retorna el coseno de un número.

exp()

Retorna el resultado de elevar el número E por un número.

floor()

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

log()

Devuelve el logaritmo neperiano de un número.

max()

Retorna el mayor de 2 números.

min()

Retorna el menor de 2 números.

pow()

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

random()

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

round()

Redondea al entero más próximo.

sin()

Devuelve el seno de un número con un ángulo en radianes.

sqrt()

Retorna la raiz cuadrada de un número.

tan()

Calcula y devuelve la tangente de un número en radianes.

Ejemplo de utilización de la clase Math

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

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

document.write ("<br>")

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

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

Podemos ver el resultado de ejecutar estas líneas de código.

Además, si deseamos profundizar en la clase Math os recomiendo leer el artículo de crear un número aleatorio. También se hace uso de la clase Math en el artículo enlace aleatorio. Todos en el Taller de Javascript.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *25/04/2002*
Disponible online en *http://desarrolloweb.com/articulos/20.php*

Clase Number en Javascript

Clase que modeliza el tipo de datos numérico.

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

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

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

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

- Si el constructor recibe un número, entonces inicializa el objeto con el número que recibe. Si recibe un número entrecomillado lo convierte a valor numérico, devolviendo también dicho número.
- Devuelve 0 en caso de que no reciba nada.
- En caso de que reciba un valor no numérico devuelve NaN, que significa "Not a Number" (No es un número).
- Si recibe false se inicializa a 0 y si recibe true se inicializa a 1.

Su funcionamiento se puede resumir en estos ejemplos.

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

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

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

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

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

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

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

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

Propiedades de la clase Number

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

NaN

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

MAX_VALUE y MIN_VALUE

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

NEGATIVE_INFINITY y POSITIVE_INFINITY

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

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

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

```
document.write("<br>")
document.write("Propiedad POSITIVE_INFINITY: " + Number.POSITIVE_INFINITY)
```

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

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *14/05/2002*
Disponible online en *http://desarrolloweb.com/articulos/20.php*

Clase Boolean en Javascript

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

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

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

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

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

Se inicializa a false

Cuando no pasas ningún valor al constructor, o si pasas una cadena vacía, el número 0 o la palabra false sin comillas.

Se inicializa a true

Cuando recibe cualquier valor entrecomillado o cualquier número distinto de 0.

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

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

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

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

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

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

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

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

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

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *14/05/2002*
Disponible online en *http://desarrolloweb.com/articulos/20.php*

Epílogos a esta primera parte del Manual de Javascript

Con estos artículos terminaremos la primera parte del manual de Javascript de DesarrolloWeb.com. Aquí veremos por dónde continuar estas explicaciones y algunos temas de interés como el control de errores en los programas.

Pausa y consejos Javascript

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

Hasta aquí hemos visto la mayor parte de la sintaxis y forma de funcionar 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 lectura 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.

> **Actualizado:** a día de hoy todavía las diferencias entre los navegadores antiguos y los nuevos es todavía más patente. Incluso, ahora que aparece el HTML 5, existen muchos otros usos donde Javascript tiene validez.

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

Consejos para hacer un código sencillo y fácil de mantener

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

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

En el resto de esta primera parte del manual de Javascript vamos a ver un par de cosas también fundamentales en el trabajo del día a día con este lenguaje, como son la detección de errores. Además, os dejamos unas referencias importantes para continuar con el aprendizaje:

- Manual de Javascript II: la segunda parte de este manual abordará usos de Javascript para control de los elementos de las páginas web.
- Videotutorial de Javascript: estamos publicando una serie de videotutoriales de Javascript que te encantarán si quieres aprender por la práctica.
- Javascript a fondo: todo lo que tiene que ver con Javascript lo encontrarás en esta sección de DesarrolloWeb.com.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *10/01/2002*
Disponible online en *http://desarrolloweb.com/articulos/20.php*

Tratamiento de errores en javascript

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

Hemos terminado ya de explicar todo el contenido de la primera parte del manual de javascript, pero aun tenemos algunas cosas importantes que tratar. En concreto en este artículo 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 en Javascript

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.

> **Nota:** Para aprender a utilizar las herramientas de detección de errores Javascript más populares, recomendamos especialmente ver el videotutorial sobre detectar errores Javascript en páginas web.

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.

Este artículo es obra de *Miguel Angel Alvarez*
Fue publicado por primera vez en *11/02/2002*
Disponible online en *http://desarrolloweb.com/articulos/20.php*

Consejos para escribir código Javascript

En este artículo puedes encontrar varios consejos bastante interesantes a la hora de programar código Javascript.

Si estas dando tus primeros pasos en Javacript 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 o en 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 amplies el código, tampoco seas extremista y hagas 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 un 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 maneras 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 unicamente las etiquetas sin necesidad de acceder al código.

Documenta el código

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

Este artículo es obra de *Manu Gutierrez*
Fue publicado por primera vez en *20/12/2007*
Disponible online en *http://desarrolloweb.com/articulos/consejos-para-escribir-codigo-javascript.html*