JAVASCRIPT

Ahora te encuentras en el inicio de la documentación que trata sobre el lenguaje Web Javascript. Durante la lectura aprenderás a hacer dinámicas tus páginas web y hacerlas mucho más atractivas para los visitantes.

Esta documentación cubrirá muchos temas, desde lo básico. Aprenderás cómo hacer animaciones, aplicaciones complejas y utilizar este lenguaje junto con HTML5, la nueva versión del famoso W3C.

Esta documentación discutirá principalmente el uso de JavaScript en un entorno de navegador Web, por lo que es esencial que sepas codificación HTML y CSS. Conocer PHP puede ser una ventaja.

Parte 1: Conceptos básicos de Javascript

Como cualquier otro lenguaje de programación, JavaScript tiene algunas características especiales: sintaxis, modelo de objetos, etc. Claramente, cualquier cosa que diferencia un lenguaje de otro. Además, descubrirás rápidamente que JavaScript es un lenguaje relativamente especial en su acercamiento a las cosas. Esta parte es esencial para cualquier principiante de programación e incluso para aquellos que ya conocen un lenguaje de programación debido a que las diferencias con otros lenguajes de programación son numerosas.

Introducción a JavaScript

Antes de entrar directamente en el núcleo de la cuestión, este capítulo te enseñará lo que Javascript, puede hacer, cuando se puede o se debe utilizar y cómo ha evolucionado desde su creación en 1995.

También vamos a discutir algunos conceptos básicos tales como las definiciones exactas de ciertos términos.

¿Qué es JavaScript?

JavaScript es un lenguaje de programación de *scripts* (secuencia de comandos) orientado a objetos. Esta descripción es un poco rudimentaria, hay varios elementos que vamos a diseccionar.

- Un lenguaje de programación

En primer lugar, un **lenguaje de programación** es un lenguaje que permite a los desarrolladores escribir código fuente que será analizado por un ordenador.

Un **desarrollador** o programador es una persona que desarrolla programas. Puede ser un profesional (un ingeniero, programador informático o analista) o un aficionado.

El **código fuente** está escrito por el desarrollador. Este es un conjunto de acciones, llamadas instrucciones, lo que permitirá dar órdenes al ordenador para operar el programa. El código fuente es algo oculto, como un motor en un automóvil está oculto, pero está ahí, y es quien asegura que el coche puede ser conducido. En el caso de un programa, es lo mismo, el código fuente rige el funcionamiento del programa.

Dependiendo del código fuente, el ordenador realiza varias acciones, como abrir un menú, iniciar una aplicación, efectuar búsquedas, en fin, todo lo que el equipo es capaz de hacer.

Hay una gran cantidad de lenguajes de programación, la mayoría se encuentran en esta página de la Wikipedia.

- *Scripts* de programación

JavaScript te permite programar *scripts*. Como se mencionó anteriormente, un lenguaje de programación es utilizado para escribir código fuente a ser analizada por un ordenador. Hay tres formas de usar el código fuente:

Lenguaje compilado como: El código fuente se da a un programa llamado compilador que lee el código fuente y lo convierte en un lenguaje que el equipo será capaz de interpretar: el lenguaje binario, es de 0 y 1. Lenguajes como C o C ++ son lenguajes compilados muy conocidos.

Lenguaje precompilado: aquí, el código fuente se compila en parte, por lo general en un código más fácil de leer para el ordenador, pero que todavía no es binario. Este código intermedio es para ser leído por lo que se llama una “Máquina Virtual", que ejecutará el código. Lenguajes como C # o Java se llaman precompilados.

Lenguaje interpretado: en este caso, no hay compilación. El código fuente se mantiene sin cambios, y si desea ejecutar este código, debemos proporcionar un intérprete que va a leer y realizar las acciones solicitadas.

Los scripts son en su mayoría interpretados. Y cuando decimos que JavaScript es un lenguaje interpretado, lo que significa que es un lenguaje interpretado. Por tanto, es necesario contar con un intérprete para ejecutar código Javascript, y el intérprete que se utiliza una frecuencia: se incluye en tu navegador de internet.

Cada navegador tiene un intérprete Javascript, que varía en función del navegador. Si está utilizando Internet Explorer, el intérprete es llamado JScript (versión 9 intérprete llamado Chakra), en Mozilla Firefox se llama SpiderMonkey y el motor V8 es el de Google Chrome.

- Lenguaje orientado a objetos

Queda una specto a analizar: **orientado a objetos**. Este concepto es bastante complicado de configurar ahora y se profundizará sobre todo después de la parte 2. Sin embargo, un lenguaje de programación orientado a objetos es un lenguaje que contiene elementos, llamados objetos y los objetos diferentes tienen características específicas y formas de uso diferente. El lenguaje proporciona objetos básicos, como imágenes, fechas, cadenas de caracteres... También es posible crear tus propios objetos para hacer la vida más fácil y obtener un código fuente más claro (fácil de leer) y una forma de programar mucho más intuitivo (lógica).

Es muy probable que no entiendas este paso si nunca ha realizado programación, pero no: comprenderás muy pronto cómo funciona todo.

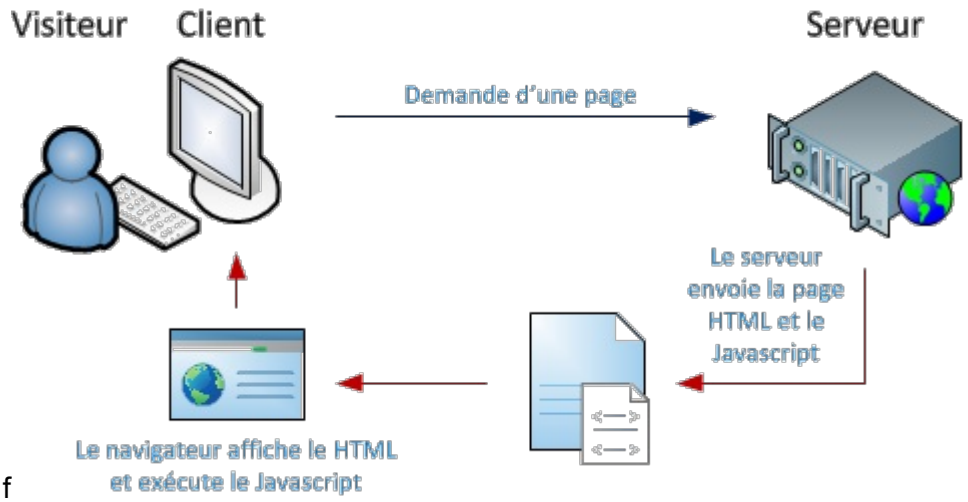
JavaScript, el lenguaje de scripts

Javascript actualmente es principalmente utilizado en internet, junto con las páginas web (HTML o XHTML). Javascript está directamente incluido en la página web (o en un archivo externo) y mejora una página HTML, añadiendo interacción del usuario, animación, ayudas a la navegación, tales como:

Mostrar / ocultar el texto;
Deslizamiento de imágenes;
Crear presentaciones de diapositivas;
Crear burbujas de información.

De JavaScript se dice que es un lenguaje del lado del cliente, es decir que los scripts son ejecutados por el navegador del usuario (cliente). Esto difiere de los llamados lenguajes de script del lado del servidor que son ejecutadas por el servidor web. Este es el caso de lenguajes como PHP.

Esto es importante porque el propósito de los scripts del lado del cliente y del lado del servidor no es el mismo. Un script del lado del servidor se encargará de "crear" la página web que se envía al navegador. Este entonces mostrará la página a continuación, ejecutará secuencias de comandos del lado del cliente como JavaScript. Un patrón que se repite en esta operación:



f

Javascript no es la Web

Si Javascript está diseñado para ser usado en conjunción con HTML, el lenguaje ha evolucionado desde entonces hacia otros destinos. Javascript es regularmente utilizado para hacer extensiones para diferentes programas, como los scripts codificados en Lua o Python.

JavaScript también se puede utilizar para construir aplicaciones. Mozilla Firefox es el ejemplo más famoso: la interfaz del navegador se crea con una especie de HTML llamado XUL y JavaScript que se utiliza para animar la interfaz. Otros programas también están basados en esta tecnología, como por ejemplo de *TomTom HOME* que se utiliza para administrar tu navegador GPS TomTom a través de tu PC.

Breve historia del lenguaje

En 1995, Brendan Eich trabajaba en *Netscape Communications Corporation*, la compañía que publicó el famoso Netscape Navigator, entonces principal competidor de Internet Explorer. Brendan desarrolló Live Script un lenguaje de script que se basa en el lenguaje Java, y que estaba destinado a ser instalado en los servidores desarrollados por Netscape. Netscape inició el desarrollo de una versión del cliente LiveScript, que pasó a llamarse JavaScript, en homenaje al lenguaje Java creado por Sun Microsystems.

En efecto, en ese momento, el lenguaje Java era cada vez más popular, y Brendan Eich, el padre de Javascript, al llamarlo JavaScript en lugar de LiveScript era una forma de publicidad de Java y JavaScript en sí. Atención, al final, estos dos lenguajes son radicalmente diferentes, no vayas a confundir Java y JavaScript, pues operan de forma diferente.

Javascript fue lanzado en diciembre de 1995 y estaba integrado en Netscape Navigator 2. El lenguaje fue tan exitoso, por lo que Microsoft desarrolló una versión similar llamada JScript, que se instalaba en Internet Explorer 3, en 1996. Netscape decidió enviar a su versión de Javascript a ECMA International (*European Computer Manufacturers Association*, la Asociación Europea

de Normalización de hoy los sistemas de información y comunicación) para que el lenguaje fuera normalizado, es decir para que se creara una referencia del lenguaje y que así puediera ser utilizado por otras personas y embebidos en otro *software*. ECMA International estandarizó el lenguaje con el nombre de ECMAScript. Desde entonces, las versiones de ECMAScript han evolucionado. La versión más conocida es utilizada en todo el mundo, es la versión ECMAScript 3, publicado en diciembre de 1999.

ECMAScript y sus derivados

ECMAScript es la línea de base en el flujo de implementaciones de referencia. Obviamente, se puede citar a Javascript, que se implementa en la mayoría de los navegadores, pero también:

JScript, que está embebido en la aplicación Internet Explorer. También es el nombre del intérprete de Internet Explorer;
JScript.NET, que se inserta en el marco de Microsoft NET.;
ActionScript, que es la implementación realizada por Adobe en Flash;
EX4 que es el desarrollo de gestión de ECMAcript de XML en el seno del intérprete JavaScript
SpiderMonkey,de Firefox.

Versiones de Javascript

Las versiones de Javascript se basan en los de la ECMAScript (que abreviaremos como ES). Por lo tanto, se encuentran:

ES 1 y ES 1, que son los inicios de Javascript;
ES 3 (publicada en diciembre de 1999), que es funcional en todos los navegadores (excepto las versiones anteriores de Internet Explorer);
ES 4, que fue abandonada debido a los grandes cambios que no fueron apreciados;
ES 5 (publicada en diciembre de 2009), que es la versión más reciente liberada;
ES 6, que se encuentra actualmente en fase de diseño.

Esta documentación cubrirá todas las versiones actualizadas.

Un logotipo desconocido

No hay imágenes oficiales para representar Javascript. Sin embargo, este logotipo se utiliza cada vez más por la comunidad, especialmente desde su introducción en EE.UU. en JSConf EU. Se puede encontrar en esta dirección en diferentes formatos.

Resumen

- JavaScript es un lenguaje de programación interpretado, es decir, que necesita un intérprete para ser ejecutado.

- JavaScript se utiliza principalmente en páginas web.
- Al igual que HTML, JavaScript es ejecutado por el navegador del usuario: se llama un de cliente, en comparación con el lado del servidor cuando el código es ejecutado por el servidor.
- Javascript está normalizado por ECMA International como el nombre *ECMAScript Language Reference*.
- Hay otros lenguajes derivadas del ECMAScript como ActionScript, EX4 o JScript.NET.
- La última versión del estándar está basado en ECMAScript 5, lanzado en 2009.

Primeros pasos en Javascript

Como se mencionó anteriormente, JavaScript es un lenguaje utilizado principalmente con el lenguaje HTML, en este capítulo se aprende cómo integrar este lenguaje en tus páginas web, descubrir su sintaxis básica y mostrar un mensaje en la pantalla del usuario.

También se encuentran al final de este capítulo algunos enlaces que pueden probablemente ser útiles durante la reproducción de esta documentación.

En cuanto al editor de texto a usar (en el que se escribe el código Javascript) es muy probable que valga el que se ha empleado para el código HTML.

Muestra un cuadro de diálogo

Hola Mundo! No se deroga la regla tradicional de que todos los tutoriales de programación comenzarán mostrando el texto "*Hello World!*", ("¡Hola Mundo!" en español) al usuario. A continuación se muestra un programa HTML simple que contiene la instrucción Javascript, situada dentro de un elemento <script>:

Código: HTML - Hola Mundo!

```
<!DOCTYPE html>
<html>
<head>
<title>Hello World!</title>
</head>
<body>
        <script>
        alert('Hello world!');
        </script>
</body>
</html>
```

Apareciendo al ejecutarlo la siguiente caja de diálogo:

Lo nuevo

En el código HTML indicado anteriormente, vemos algunas nuevas características. En primer lugar, un elemento <script> está presente: es él quien tiene el código javascript de la siguiente manera:

Código: JavaScript

```
alert ('Hello world');
```

Es una declaración, es decir, una orden, o más bien una acción que el equipo tendrá que hacer. los lenguajes de programación consisten en una secuencia de instrucciones que, colocados de extremo a extremo, permiten obtener un programa o un script completo. En este ejemplo, no es una instrucción: se llama a la función alerta.

El cuadro de diálogo de alert ()

alert () es una declaración simple, llamada función, que muestra un cuadro de diálogo que contiene un mensaje. este mensaje se coloca entre comillas, entre los paréntesis de la function alert ().

Sintaxis de Javascript

Instrucción

La sintaxis de Javascript no es complicada. Generalmente, las instrucciones deben estar separadas por un punto y coma que se coloca al final de cada instrucción:

Código: JavaScript

```
sentencia_1;
sentencia_2;
sentencia_3;
```

En realidad, el punto y coma no es necesario si la instrucción siguiente está en la línea posterior como en nuestro ejemplo. Sin embargo, si escriben varias instrucciones en una sola línea, como en el siguiente ejemplo, el punto y coma es obligatorio. Si lo virtual punto

Código JavaScript

```
sentencia_1; sentencia_2
sentencia_3
```

Compresión de scripts

Algunas secuencias de comandos están disponibles en un formato comprimido, es decir, todo el código se escribe como secuencia, no hay retornos de línea. Esto reduce considerablemente el tamaño de una secuencia de comandos y sirve para asegurar que la página se carga más rápidamente. Existen programas para "comprimir" el código Javascript. Pero si has olvidado un solo punto y coma el código comprimido no va a funcionar porque las instrucciones no están debidamente separadas. También es una de las las razones por las que siempre se pone el punto y coma al final de la instrucción.

Espacios

Javascript no es sensible a los espacios. Esto significa que puedes alinear las instrucciones que quieras, siempre que no interfiera con la secuencia de comandos. Por ejemplo, esto es correcto:

Código JavaScript

```
instruccion_1;
        instruccion_1_1;
        instruccion_1_2;
instruccion_2;      instruccion_3;
```

Sangría y presentación

La sangría en la programación informática, es una manera de estructurar el código para hacerlo más legible. Las instrucciones son priorizadas en varios niveles y espacios de usos o lengüetas para desplazar a la derecha y crear una jerarquía. Un ejemplo de código sangrado:

Código: JavaScript

```
function interruptor(elemID) {
        var elem = document.getElementById(elemID);

        if (elem.style.display == 'block') {
```

```
        elem.style.display = 'none';
        } else {
        elem.style.display = 'block';

        }
}
```

La presentación de los códigos también es importante, es como si estuvieras escribiendo una carta: no hay reglas predefinidas para escribir cartas, por lo que tendrás que hacer arreglos para organizar tu código para mostrarlo claro. En el código sangrado mostrado anteriormente, se puede ver que hay espacios para “airear” todo el código y sólo hay una declaración por línea (salvo if else, pero ya llegaremos a eso más adelante). Algunos desarrolladores escriben su código como este:

Código: JavaScript

```
function interruptor(elemID){
        var elem=document.getElementById(elemID);
        if(elem.style.display=='block'){
                elem.style.display='none';
        }else{elem.style.display='block';}
}
```

Comentarios

Los comentarios son anotaciones realizadas por el desarrollador para explicar el funcionamiento de un script, una instrucción o incluso un grupo de instrucciones. Los comentarios no interfieren con la ejecución de un script.

Hay dos tipos de comentarios: los de fin de línea y los multilínea.

Comentarios de fin de línea. Se utilizan para comentar una instrucción. Comienza con dos barras de división:

Código: JavaScript

```
sentencia_1 / / Esta es mi primera instrucción
sentencia_2;
/ / La tercera declaración es la siguiente:
sentencia_3;
```

El texto colocado en un comentario se ignora cuando se ejecuta un script, lo que significa que puedes poner un comentario, incluso en una instrucción (que, obviamente, no se ejecutará):

Código: JavaScript

```
sentencia_1 / / Esta es mi primera instrucción
sentencia_2;
/ / La tercera declaración da problemas, la cancelo temporalmente
/ / sentencia_3;
```

Comentarios multilínea. Este tipo permite saltos de línea. Un comentario multilínea comienza con / * y termina con * /:

Código: JavaScript

```
/ * Este script consta de tres pasos:
- Instrucción uno está haciendo algo
- Instrucción dos para otra cosa
- Instrucción tres que pone fin a la secuencia de comandos
* /
sentencia_1;
sentencia_2;
sentencia_3 / / Fin del script
```

Ten en cuenta que un comentario de varias líneas se pueden mostrar en una sola línea:

Código: JavaScript

```
sentencia_1 / * Esta es mi primera instrucción * /
sentencia_2;
```

Funciones

En el ejemplo de ¡Hola mundo!, Se utilizó la función alert (). Discutiremos en detalle la funciones de trabajo, en los capítulos siguientes, necesitarás saber el resumen de la sintaxis.

Una función consiste en dos partes: su nombre, seguido por un par de paréntesis (una apertura y un cierre):

Código: JavaScript

```
myFunction () / / "function"  quiere decir "función" en Inglés
```

Entre paréntesis se indican los argumentos que también se llaman parámetros. Estos contienen los valores que se pasan a la función. En el caso de ¡Hola mundo!, Son las palabras "¡Hola, mundo! " lo que se transfieren como parámetros:

Código: JavaScript

```
alert ('Hola mundo!');
```

Dónde colocar el código en la página

Los códigos JavaScript son insertados a través del elemento <script>. Este elemento tiene un atributo de tipo que se utiliza para indicar el tipo de lenguaje que vamos a utilizar. En nuestro caso, es JavaScript, pero podría ser otra cosa, como por ejemplo VBScript, aunque esto es extremadamente raro.

En HTML 4 y XHTML 1.x, el tipo de atributo es obligatorio. En contraste, en HTML5, no lo es. Esta es la razón por la que los ejemplos aquí mostrados, no incluirán este atributo. Si no estás usando HTML5, sabemos que el atributo de tipo toma como valor text / javascript, que es en realidad el tipo MIME de un código Javascript.

El tipo MIME es un identificador que describe un formato de datos. Aquí, con text / javascript, se trata de datos de texto y JavaScript.

Javascript "en la página"

Para situar el código JavaScript directamente en una página web, nada más simple, siguiendo el ejemplo de ¡Hola, mundo!: se coloca el código en el elemento <script>:

Código: HTML

```
<!DOCTYPE html>
<html>
	<head>
		<title>¡Hola Mundo!</title>
	</head>
<body>
	<script>
		alert('¡Hola Mundo!');
	</script>
</body>
</html>
```

Encuadramiento de caracteres reservados

Si utilizas el estándar HTML 4.01 y XHTML 1.x a menudo, es necesario utilizar comentarios de encuadramiento para que la página cumpla con las normas. Si por contra, como en este supuesto, se utiliza el estándar HTML5, los comentarios de encuadramiento son inútiles.

Los comentarios de encuadramiento se utilizan para aislar el código Javascript para que el Validator W3C (*World Wide Web Consortium*) no los interprete. Por ejemplo, si tu código Javascript contiene galones < y >, el validador cree que HTML no está cerrado, por lo que anularía la página. Esto no es grave en sí mismo, sino que una página sin errores, siempre es preferible.

Los comentarios de encuadramiento son como los comentarios HTML:

Código: HTML

```
<body>
<script>
        <!--
        valor_1 > valor_2;
        //-->
</script>
</body>
```

Javascript externo

Es posible, y conveniente escribir el código JavaScript en un archivo externo con la extensión. Js. Este archivo se llama desde la página web mediante el elemento <script> y su atributo src que contiene la dirección URL del archivo. js.

He aquí un pequeño ejemplo:

Código: JavaScript - contenido de ficheros hola.js

```
alert ('¡Hola mundo!');
```

Código: HTML - Página Web

```
<! DOCTYPE html>
<html>
        <head>
        <title>¡Hola mundo!!</title>
        </head>
<body>
        <script src="hola.js"></script>
```

```
</body>
</html>
```

Se supone que el archivo hola.js se encuentra en el mismo directorio que el programa en HTML.

Posición del elemento <script>

La mayoría de los cursos de Javascript, y ejemplos, muestran la necesidad de colocar el elemento <script> dentro de <head> cuando se utiliza para cargar un archivo javascript. Eso es correcto, sí, pero hay mejoras.

Una página web es leída por el navegador de forma lineal, es decir, en primer lugar lee <head>, después los elementos de <body> uno después del otro. Si se llama a un archivo JavaScript desde el principio de la carga de la página, el navegador por lo tanto cargará este archivo, y si es grande, la carga de la página se desacelerará. Esto es normal, ya que el navegador cargará el archivo antes de empezar a mostrar el contenido de la página.

Para superar este problema, es conveniente colocar los elementos <script> justo antes de cerrar <body> (algunos navegadores modernos lo hacen automáticamente) como el siguiente ejemplo:

Código: HTML

```
<!DOCTYPE html>
<html>
<head>
	<title>¡Hola Mundo!</title>
</head>
<body>
	<p>
	<!--
	Contenido de la página Web
	...
	-->
	</p>
<script>
// Un poco de código JavaScript...
	</script>
	<script src="hola.js"></script>
</body>
</html>
```

Resumen

- Las instrucciones deben estar separadas por un punto y coma.

- Un código JavaScript bien presentado es más legible y más fácil de editar.

- Es posible incluir comentarios con los caracteres / /, / * y / *.

- Los códigos Javascript son colocados en un elemento de script.

- Es posible incluir un archivo JavaScript con el atributo src del elemento <script>.

Variables

A lo largo de la lectura, descubrirás el uso de variables, los principales tipos que pueden contener y sobre todo la forma de hacer tus primeros cálculos. También se presenta la concatenación y los tipos de conversión. Y, por último, una parte importante de este capítulo trata el uso de una nueva característica que permite interactuar con el usuario.

¿Qué es una variable?

En pocas palabras, una variable es un espacio de almacenamiento en un ordenador para grabar cualquier tipo de datos como una cadena de caracteres, un valor numérico o estructuras un poco más específicas.

Declarar una variable

En primer lugar, ¿qué significa "declarar una variable" significa? Se trata simplemente de espacio de almacenamiento de reserva en memoria, nada más. Una vez que se declara la variable, puedes comenzar a almacenar datos sin problema.

Para declarar una variable, primero debes encontrar un nombre. Es importante destacar que el nombre de una variable puede contener sólo caracteres alfanuméricos, es decir, letras de la A a la Z y números del 0 al 9, guión bajo (_) y dólar ($) también son aceptados. Algo más: el nombre de la variable no puede comenzar con un número y no puede consistir únicamente de palabras clave utilizadas por Javascript. Por ejemplo, no se puede crear una variable llamada var porque encontrarás que esta palabra clave ya está en uso, sin embargo, puedes crear una variable llamada var_.

En las palabras clave utilizadas por JavaScript, se pueden llamar "palabras reservadas", simplemente porque no tienes el derecho de usarlos como nombres de variables. Encontrarás en esta página (en Inglés) todas las palabras reservadas en Javascript.

Para declarar una variable, simplemente hay que escribir la siguiente línea:

Código: JavScript

```
var miVariable;
```

JavaScript es un lenguaje sensible en las denominaciones, ten cuidado de no confundir las mayúsculas y minúsculas. En el siguiente ejemplo, tenemos tres variables diferentes declaradas:

Código: JavaScript

```
var miVariable;
var mivariable;
var MIVARIABLE;
```

La palabra clave var está presente para indicar que se declara una variable. Una vez que se declara, se puede almacenar lo que quieras:

Código: JavaScript

```
var miVariable;
miVariable = 2;
```

El signo = se utiliza para asignar un valor a la variable, aquí le hemos asignado el número 2. Cuando das un valor a una variable, decimos que se trata de una asignación, ya que asigna un valor a la variable.

Es posible simplificar el código en una sola línea:

Código: JavaScript

```
var miVariable = 5.5 / / Como puedes ver, los números decimales se separan con un punto
```

Del mismo modo, puedes declarar y asignar variables en una sola línea:

Código: JavaScript

```
var miVariable1, miVariable2 = 4, miVariable3;
```

Aquí hemos declarado tres variables en una fila, pero sólo la segunda tiene valor asignado .

Y la última posibilidad, que puede ser útil de vez en cuando:

Código: JavaScript

```
var miVariable1, miVariable2;
miVariable1 = miVariable2 = 2;
```

Ambas variables ahora contienen el mismo número 2. Puedes hacer lo mismo con tantas variables como desees.

Tipos de variables

A diferencia de muchos lenguajes, JavaScript es un lenguaje de tipado *dinámicamente*. Esto significa, generalmente, que cualquier declaración de variables se hace con la palabra clave var independientemente de su contenido, mientras que en otros lenguajes, como el C, es necesario especificar el tipo de contenido que tendrá que contener la variable.

En Javascript, nuestras variables son tipadas dinámicamente, lo que significa que puedes asignarle texto primero y luego borrarlo y poner un número cualquiera y sin restricciones.

Vamos a empezar por ver cuáles son los tres tipos principales de Javascript:

El **tipo numérico (**número): representa cualquier número, ya sea un entero, un número negativo, en notación científica, etc. En pocas palabras, este es el tipo de números.

Para asignar un tipo numérico a una variable, sólo tienes que escribir el número var numero = 5; Al igual que muchos lenguajes, JavaScript reconoce varios formatos para los números, como por ejemplo escribir var numero = 5.5 o en notación científica var número = 3.65 e5, o escribir el número hexadecimal, var numero= 0x391; En resumen, hay diferentes maneras de escribir los valores numéricos.

Cadenas de caracteres (alias *string*): Este tipo representa texto. Puede asignarse de dos maneras diferentes.

Código: JavaScript

```
var text1 = "Mi primer texto" / / Con comillas
var text2 = 'Mi segundo mensaje' / / Con apóstrofos
```

Es importante tener en cuenta que si escribes miVariable var = '2 ', el tipo de esta variable es
cadena de caracteres y no un tipo numérico.

Otro punto importante, si usas apóstrofos para "enmarcar" el texto y deseas utilizar apóstrofos en el texto mismo, entonces tienes que "escapar" de los apóstrofos como se indica seguidamente:

Código: JavaScript

```
var text = 'Esto  \' es algo ';
```

¿Por qué esto? Porque si no cancelas tu apóstrofo, Javascript cree que el texto se detiene en el apóstrofo contenido en la palabra "es". Ten en cuenta que este problema es idéntico al de las comillas.
En nuestro caso, solemos utilizar apóstrofos pero cuando el texto los contiene también entonces las comillas pueden ser muy útiles.

Booleanos (booleano): son un tipo particular de que se tratará más adelante. Mientras tanto, en pocas palabras, un tipo booleano permite dos estados verdadero o falso. Estos dos estados se puede escribir como sigue:

Código: JavaScript

```
var EsVerdader = true;
var EsFalso = false;
```

Hay otros tipos, que se considerarán cuando sea necesario.

Prueba de la existencia de variables con typeof

Puede ser que tengas en alguna ocasión la necesidad de probar la existencia de una variable o comprobar su tipo. En tales situaciones, la instrucción typeof es muy útil, así es como se usa:

Código: JavaScript

```
var numero = 2;
alert (typeof numero ) / / Muestra: « number »

var text = "mitexto";
alert (typeof mitexto) / / Muestra: « string »

var aBoolean = false;
alert (typeof aBoolean) / / Muestra: « boolean »
```

Y ahora cómo probar la existencia de una variable:

Código: JavaScript

```
alert (typeof nada) / / Muestra: « undefined »
```

Es un tipo de variable muy importante. Si la instrucción typeof devuelve undefined es que la variable es inexistente o está declarada pero no contiene nada.

Operadores aritméticos

Ahora que se declara una variable y se asignar un valor, podemos comenzar la sección sobre los operadores aritméticos. Se verá más adelante que hay varios tipos de operadores, pero por ahora nos centraremos exclusivamente en los operadores aritméticos. Estos son la base para todos los cálculos y son cinco.

Operador	Símbolo
suma	+
sustracción	-
multiplicación	*
división	/
módulo	%

El último operador, módulo, es simplemente el resto de una división. Por ejemplo, si se divide 5 entre 2 se tiene resto 1, que es el módulo.

Algunos cálculos sencillos

Programar el cálculo es casi tan fácil como en una calculadora, por ejemplo:

Código: JavaScript

```
var resultado = 3 + 2;
alert (resultado) / / Muestra « 5 »
```

Así que puedes hacer cálculos con dos números, es bueno, pero con dos variables que contienen números en sí es más útil:

Código: JavaScript

```
var  número1=3, número2 = 2, resultado;
resultado = numero1 * numero2;
alert (resultado) / / Muestra: « 6 »
```

Podemos ir aún más lejos al escribirlo como cálculos con operadores múltiples y variables:

Código: JavaScript

```
var divisor = 3, resultado1, resultado2, resultado3;
resultado1 = (16 + 8) / 2 - 2; / / 10
resultado2 = resultado1 / divisor;
resultado3 = resultado1 % divisor;
alerta (resultado2) / / Resultado de la división: 3,33
```

```
alerta (resultado3) / / Resto de la división: 1
```

Notarás que usamos paréntesis para el cálculo de la variable resultado1. Se utilizan como en matemáticas: el navegador que primero calcula 16 + 8 y divide el resultado por 2.

Simplificar los cálculos

A veces tendrás que escribir cosas como:

Código: JavaScript

```
var numero = 3;
numero = numero + 5;
alert (numero) / / Muestra: « 8 »
```

Esto no es particularmente largo o complicado de hacer, pero puede convertirse rápidamente en un proceso de enormes proporciones, existe una manera de agregar un número a una variable:

Código: JavaScript

```
var numero = 3;
numero + = 5;
alert (numero) / / Muestra: « 8 »
```

Este código tiene el mismo efecto que el anterior, pero es más rápido de escribir. Ten en cuenta que esto no sólo se aplica a las sumas, pero trabaja con los otros operadores aritméticos:

```
+ =
- =
* =
/ =
% =
```

Introducción a la concatenación y conversión de tipos

Algunos operadores se han omitido previamente. Toma el operador +, además de las sumas, se puede hacer lo que se conoce como concatenación entre cadenas.

Concatenación

La concatenación es añadir una cadena al final de otra, como en este ejemplo:

Código: JavaScript

```
var hola= 'Hola', nombre = 'tu', resultado;
resultado = hola + nombre;
alert (resultado) / / muestra: « Holatu »
```

En este ejemplo se muestra la frase "Holatu". Te darás cuenta de que no hay espacio entre las dos palabras, de hecho, es la concatenación de lo que se definió en las variables. Si quieres un espacio, debes agregar un espacio en una variable, como var hola = 'Hola ';

Si se recuerda el ejemplo previo de adición, que se expresaba +=, se puede actuar de forma análoga con cadenas de caracteres:

Código: JavaScript

```
var text = "Hola ";
texto + = 'tu';
alert (texto) / / Muestra « Hola tu ».
```

Interactuar con el usuario

La concatenación es un buen momento para introducir la primera interacción con el usuario a través de la función prompt (). He aquí cómo se usa:

Código : JavaScript

```
var NombreUsuario = prompt('Introduce nombre:');
alert(NombreUsuario); // Muestra el nombre introducido
```

fig pag 29

La función prompt () se utiliza como alert (), pero tiene una pequeña particularidad. Devuelve lo que el usuario escribió bajo una cadena de caracteres, es por eso que escribió esto:

Código: JavaScript

```
var texto = prompt ('Entra algo:');
```

Así, el texto escrito por el usuario terminará almacenado directamente en el texto variable.

Ahora podemos tratar de saludar a nuestros visitantes:

Código: JavaScript

```
var inicio = 'Hola', nombre, final = ' !', resultado;
nombre= prompt ("¿Cuál es tu nombre? ');
resultado = inicio + nombre + final
alert (resultado);
```

Ten en cuenta que en nuestro caso concatenamos cadenas de caracteres, pero se pueden concatenar una cadena y un número de la misma manera:

Código : JavaScript

```
var text o= 'Un nombre : ', numero= 42, resultado;
resultado= texto + numero;
alert(resultado); // Muestra: « Un nombre : 42 »
```

Conversión de una cadena de caracteres en número

Ahora trataremos de hacer una adición con números proporcionados por el usuario:

Código: JavaScript

```
var primero, segundo, resultado;
primero = prompt ('Introduzca el primer número: ");
segundo = prompt ('Introduzca el segundo número:');
resultado = primero + segundo;
alert (resultado);
```

Si has probado este código, te habrás dado cuenta de que hay un problema. Supongamos que has escrito dos veces el dígito 1, el resultado será 11 ... ¿Por qué? La razón se ha escrito unas líneas más arriba:

> Se devuelve al usuario lo que ha escrito bajo forma de una cadena de caracteres [...]

El problema es que todo lo que se escribe en el campo de texto prompt () se recupera como una cadena de caracteres, aunque sea un número. Por lo tanto, si se utiliza el operador +, no será una suma sino una concatenación.
Se ha de efectuar una conversión. El concepto es simple: convertir la cadena en un número. Para ello, necesitarás la función parseInt () que se utiliza de esta manera:

Código: JavaScript

```
var texto = '1313', numero;
numero = parseInt(texto);
```

```
alert(typeof numero); // Muestra : « numero »
alert(numero); // Muestra : « 1313 »
```

Ahora que ya sabes cómo usarlo, vamos a ser capaces de adaptar nuestro código:

Código: JavaScript

```
var primero, segundo, resultado;
primero = prompt ('Ingrese el primer número: ");
segundo = prompt ('Ingrese el segundo número:');
resultado = parseInt (primero) + parseInt (segundo);
alert (resultado);
```

Ahora, si escribes dos veces la cifra 1, el resultado es 2.

Conversión de un número en una cadena

Para concluir esta sección, veremos cómo convertir un número en una cadena. Ya es posible concatenar un número y una cadena sin conversión, pero no dos números, tal como se les añadió debido a la utilización de +. Por lo tanto necesito convertir un número en cadena.

Código: JavaScript

```
var  texto, numero1 = 4, numero2 = 2;
text = numero1 + ' ' + numero2;
alert (texto) / / Muestra: « 42 »
```

¿Qué hemos hecho? Acabamos de añadir una cadena vacía entre los dos números, lo que tiene por efecto convertirlos en cadenas. Hay una solución un poco menos arcaica que añadir una cadena vacía, la encontrarás más adelante.

Resumen

- Una variable es una manera de almacenar un valor.

- Usamos la palabra clave var para declarar una variable, y usamos = para asignar un valor a la variable.

- Las variables se escriben de forma dinámica, lo que significa que no es necesario especificar el tipo de contenido que la variable contendrá.

- Con diferentes operadores, podemos hacer las transacciones entre las variables.

- El operador + concatena cadenas de caracteres, es decir, de inicio a fin.

- La función prompt() permite interactuar con el usuario.

Condicionales

En el capítulo anterior se aprendió cómo crear y modificar variables. Todavía se siente un tanto limitados nuestros códigos. En este capítulo, descubrirás las condiciones de todo tipo y especialmente darse cuenta de que las posibilidades del código serán mucho más abiertas porque las condicionales afectarán directamente cómo el código va a reaccionar a ciertos criterios.

En la mayoría de las condicionales, también se podrá profundizar mediante un tipo famoso de variable: boolean.

La base de cualquier condición: booleana

En este apartado, vamos a discutir las condicionales, pero para eso primero tenemos que volver a un tipo de variable que hemos mencionado en el capítulo anterior: booleanas.

¿Para qué van a servir? Para obtener un resultado como verdadero (*true*) o falso (*false*) cuando se verifica una condición. Para aquellos que se preguntan el significado , una condición es una especie de "*test*" para comprobar que una variable contiene un cierto valor. Por supuesto, las comparaciones no se limitan sólo a las variables, pero por el momento lo vamos a considerar más que suficiente para comenzar.

En primer lugar, ¿cuáles son las condiciones establecidas? Valores a ensayar dos tipos de operadores: uno lógico y el otro de comparación.

Los operadores de comparación

Como su nombre indica, estos operadores pueden realizar comparaciones entre diferentes valores entre ellos. En total, hay muchos, ocho, aquí están:

Operador	Significado
=	Igual a
!=	Diferente a
===	Contenido y tipo igual a
!==	Contenido o tipo diferente de
>	Mayor que
>=	Mayor o igual que

<	Menor que
<=	Menor o igual que

No vamos a hacer un ejemplo para cada uno de ellos, pero por lo menos te mostraremos cómo utilizarlos para que puedas probar el otro:s

Código: JavaScript

```
var numero1 = 2, numero2 = 2, numero3 = 4, resultado;

resultado = numero1 == numero2 / / En lugar de un único valor, se han escrito dos con el operador de comparación entre ellos
alert (resultado) / / Muestra "true", si la condición se verifica porque las dos variables contienen el mismo valor

resultado = numero1 == numero3;
alert (resultado) / / Muestra "false", la condición no es verificada porque 2 es distinto de 4

resultado = numero1 < numero3;
alert (resultado) / / Muestra "true", la condición es verdadera porque 2 es inferior a 4
```

Como se puede ver, el concepto no es muy complicado, simplemente escribe dos valores con el operador de comparación deseado entre los dos y devuelve un booleano. Si esto es true (verdadero), entonces la condición está verificada, si es false (falsa), entonces no.

De estos ocho operadores, dos de ellos pueden ser difíciles de entender para un principiante: se trata === y ! ==. de modo que se indica cómo trabajan con algunos ejemplos:

Código: JavaScript

```
var numero = 4, text = '4 ', resultado;
resultado = numero == texto;
alert (resultado) / / Muestra "verdadero", mientras que "número" es un
número y el "texto" de una cadena de caracteres
resultado = numero === texto;
alert (resultado) / / Muestra "false" porque este operador también compara
tipos de variables en adición a sus valores
```

Los condiciones "normales" hacen conversiones de tipos para verificar las igualdades de modo que si se quiere diferenciar el número 4 en una cadena de caracteres que contiene el número 4 entonces tienes que utilizar la igualdad triple ===.

Para todos los operadores de comparación, tienes todas las herramientas que se necesitan para hacer experimentos.

Operadores lógicos

¿Por qué estos operadores se denominan como "lógicos"? Puesto que funcionan con el mismo principio como una tabla electrónica de verdad. Antes de describir su funcionamiento, lo primero que debes hacer es una lista, son tres:

Operador	Tipo de lógica	Utilización
&&	Y	valor1 && valor2
||	O	valor1 || valor2
!	NO	!valor

- Operador Y (AND)

Este operador satisface la condición true cuando todos los valores que se pasan al mismo son verdaderos. Si uno de ellos es false, entonces la condición no será verificada. Por ejemplo:

Código: JavaScript

```
var resultado = true && true;
alert (resultado) / / Muestra "verdadero"

resultado = true && false;
alert (resultado) / / Muestra: "false"

resultado = false && false;
alert (resultado) / / Muestra: "false"
```

- Operador O (OR)

Este operador es más "flexible" porque devuelve true (verdadero) si uno de los valores que contiene es verdadero, no importan otros valores. Por ejemplo:

Código: JavaScript

```
var resultado = true | | verdadero;
alert (resultado) / / Muestra "verdadero"

resultado = true | | false;
alert (resultado) / / Muestra "verdadero"

resultado = false | | false;
alert (resultado) / / Muestra: "false"
```

- El operador NO (NOT)

Este operador se diferencia de los otros dos porque necesita sólo un valor. Se le llama "NO", porque su función es la de invertir el valor que se le pasa y se convierte verdadero en falso y viceversa. Por ejemplo:

Código: JavaScript

```
var resultado = false;

resultado =! resultado / / se almacena en "resultado" el inverso de "resultado" ello, es posible
alert (resultado) / / Muestra "true" porque queríamos el opuesto de "false"

resultado = !resultado;
alert (resultado) / / Muestra "false", ya que se invirtió de nuevo, como resultado, "se cambia de "true" a "false"
```

- Combinación de operadores

Estamos casi al final de la sección sobre valores booleanos, no te preocupes, será más fácil el resto del capítulo. Sin embargo, antes de continuar, debes asegurarte de que entiendes que todos los operadores que acabas de descubrir puedes combinarlos.

En primer lugar un breve resumen (leer atentamente): los operadores de comparación toma dos valores de entrada y devuelven un booleano, mientras que los operadores lógicos aceptar la entrada del múltiples booleanos y devuelven un booleano. Habrás entendido que entonces puedes acoplar los valores de salida de los operadores de comparación con los valores de entrada de los operadores lógicos. Por ejemplo:

Código: JavaScript

```
var condicion1, resultado, condicion2;
```

```
condicion1 = 2> 8 / / false
condicion2 = 8> 2 / / true
resultado = condicion1 && condicion2;
alert (resultado) / / Muestra "false"
```

Por supuesto, es posible acortar el código si se combinan en una sola línea, todas las condiciones:

Código: JavaScript

```
var resultado = 2> 8 && 8> 2;
alert (resultado) / / Muestra "false"
```

Condición "if else"

Finalmente se discuten las condiciones. O, más precisamente, las estructuras condicionales, ahora podemos escribir la palabra "Condición" que todavía será más rápido para escribir y leer.

Por encima de todo, ten en cuenta que hay tres tipos de condiciones, vamos a empezar con la condición if else que se utiliza con frecuencia.

Pero ¿no hemos visto antes de los operadores condicionales para obtener un resultado? De hecho, podemos obtener el resultado como un valor lógico, pero eso es todo. Ahora, sería bueno que el resultado pudiera afectar a la ejecución del código. De inmediato voy a entrar en el núcleo de la cuestión con un ejemplo muy bueno y simple:

Código: JavaScript

```
if (true) {
alert ("Este mensaje se muestra bien.");
}
if (false) {
alert ("No hay necesidad de insistir, este mensaje no se mostrará.");
}
```

En primer lugar, veamos de qué está constituida una condición:

- De la estructura condicional if;
- Paréntesis que contienen la condición de analizar, o más precisamente el valor booleano devuelto por los operadores condicionales;
- Llaves que definen la porción de código que se ejecutará si la condición es verdadera. Nótese que colocamos aquí la primera llave al final de la primera línea de condición, pero puede colocarse más o menos como se desee (por

debajo, por ejemplo).

Como se puede ver, el código de condición se ejecuta si el booleano recibido es true (verdadero), mientras que false (falso) evita la ejecución de código.

Y en vista de que los operadores condicionales reenvían booleanos, vamos a ser capaces de usarlos directamente en nuestras condiciones:

Código: JavaScript

```
if (2 <8 && 8> = 4) {/ / Esta condición devuelve "true",  entonces el código se ejecuta
alert ('La condición está verificada.');
}
if (2> 8 | | 8 <= 4) {/ / Esta condición devuelve "false", el código no se ejecuta
alert ("La condición no se verifica, pero nunca lo sabrá, pues el código no se está ejecutando ").;
}
```

Como se puede constatar, antes de que descompongamos todas las etapas de una condición en varias variables, recomendamos ponerla en una sola línea, así será más rápida de escribir y más fácil de leer.

Función confirm ()

Vamos a aprender cómo usar una característica muy útil: confirm (). Su uso es sencillo: se pasa un parámetro que es una cadena que se mostrará en pantalla y lo vuelve a booleano dependiendo de la acción del usuario,. Por ejemplo:

Código: JavaScript

```
if (confirm ('¿Desea ejecutar el código javascript de esta página?
')) {
alert ('El código se ha ejecutado bien')
```

El código se ejecuta cuando se hace clic en el botón Aceptar y no se ejecuta cuando se haga clic en el botón Cancelar (ambos botones aparecerían en pantalla, junto con la pregunta de si se desea usar el código JavaScript). En resumen: en el primer caso, la función devuelve true y en el segundo caso, devuelve false. Lo que hace que sea una característica muy conveniente para el uso con las condiciones.

Estructura else para decir "si no"

Ahora supón que deseas ejecutar código después de comprobar una condición y ejecutar otro código si no resulta verificado. Es posible ejecutar dos condiciones, pero existe una solución mucho más simple, la estructura else:

Código: JavaScript

```
if (confirm ('Para acceder a este sitio usted debe tener 18 años o más,
haga clic en "OK" si e's el caso.')) {
alert ('Será redirigido al sitio.');
}
else {
alert ("Lo siento, usted no tiene acceso a esta página.");
}
```

Como se puede ver, la estructura puede ejecutar código si la condición no se ha verificado, y 'rápidamente te darás cuenta de que será muy útil en muchas ocasiones.

Acerca de la forma de indentar la estructura if else, se recomienda seguir los siguientes pasos:

Código: JavaScript

```
if (/ * condición * /) {
        / / código ...
} else {
        / / código ...
}
```

Estructura else para decir "si no si"

Ya sabes ejecutar código si una condición es verdadera y si no es verdad, era pero sería bueno saber operar como sigue:

- Una primera condición debe analizarse;
- Una segunda condición está presente y se pondrá a prueba si la primero falla;
- Y si ninguna condición se verifica (es verdadera), la estructura else es entonces la que actúa.

Este tipo de estructura es muy útil para verificar varias condiciones a la vez y ejecutar su código correspondiente. La estructura else if lo permite, por ejemplo:

Código: JavaScript

```
var suelo = parseInt (prompt ("Escriba el piso, donde el ascensor debería parar (-2 a 30): "));
if (piso == 0) {
alert ('Usted ya está en la planta baja.');
} else if (-2 <= suelo && piso <= 30) {
alert ("Dirección a la planta nº" + suelo +'!');
else {}
alert ("El piso especificado no existe.");
}
```

Ten en cuenta que la estructura else if se puede usar repetidamente, lo único que se necesita para trabajar es que debe tener delante de ella una estructura if.

La condición "switch"

Hemos estudiado el funcionamiento de la condición if else que es muy útil en muchos casos, sin embargo, no es muy conveniente para cada caso, y es aquí donde es importante un “interruptor”, que se logra con la instrucción switch.

Un ejemplo: tenemos un mueble con cuatro cajones, cada uno conteniendo diferentes objetos, y es necesario que el usuario pueda conocer el contenido de cada cajón, que ha sido cifrado. Si lo hiciéramos con if else sería muy largo y tedioso:

Código: JavaScript

```
var cajon = parseInt (prompt ('Elegir el cajón abierto (1-4): '));
if (cajon == 1) {
alert ('Contiene varias herramientas de dibujo: papel, lápices, etc. ')
} else if (cajon == 2) {
alert ('Contenido hardware: cables, componentes, etc. ')
} else if (cajon == 3) {
alert ('¿Ah, el cajón está cerrado malo?');
} else if (cajon == 4) {
alert ('Contiene la ropa: camisas, pantalones, etc. ')
Else {}
alert ("La noticia del día: el gabinete contiene sólo cuatro cajones y
hasta que se demuestre lo contrario, los cajones negativos no existen ").;
}
```

Es largo, ¿verdad? Además de que no es muy adecuado para lo que quieres hacer. El mayor problema es tener que volver a escribir siempre la condición, pero con switch es un poco más fácil:

Código: JavaScript

```
var cajon = parseInt (prompt ('Elegir el cajón abierto (1-4): '));

switch (cajon) {
case 1:
alert ('Contiene varias herramientas de dibujo: papel, lápices, etc. ')
break;

case 2:
alert ('Contenido hardware: cables, componentes, etc. ')
break;

case 3:
alert ('¿Ah, el cajón está cerrado malo?');
break;

case 4:
alert ('Contiene ropa: camisas, pantalones, etc. ')
break;

default:
alert ("La noticia del día: el gabinete contiene sólo cuatro cajones
y, hasta que se demuestre lo contrario, los cajones negativos no
existen");
}
```

Como se puede ver, el código no es especialmente corto, pero se organiza mejor y por lo tanto es más comprensible. Ahora el detalle de cómo funciona:

Escribimos la palabra clave switch seguida de la variable a analizar entre paréntesis y un par de llaves;

Entre las llaves se encuentra en todos los casos la variable definida por la palabra clave case seguida del valor se debe tener en cuenta (esto puede ser un número, o también texto) y dos puntos;

Todo lo que sigue a los dos puntos de case se ejecutará si la variable analizada por switch contiene el valor de case;

Al final de case está la sentencia break para "romper" el switch y así evitar la ejecución del resto del código que contiene;

Y finalmente escribir la palabra clave default seguida de dos puntos. El código que sigue a esta instrucción será ejecutado si ninguno de los casos anteriores se han ejecutado. Advertencia, esta parte es opcional, no es necesario para la integración con el código.

En general, no tendrás problemas para entender el funcionamiento de switch, sin embargo la declaración break puede ser un problema, te invitamos a probar el código sin esta instrucción.

¿Empiezas a entender el problema? Sin la instrucción break se ejecuta todo el código contenido en switch a partir del case que has seleccionado. Por lo tanto, si eliges el cajón nº 2 es como si se ejecutara este código:

Código: JavaScript

```
alert ('Contenido hardware: cables, componentes, etc. ')
alert ('¿Ah el cajón está cerrado malo?');
alert ('Contiene la ropa:. camisas, pantalones, etc');
alert ("La noticia del día: el gabinete contiene sólo cuatro cajones y hasta que se demuestre lo contrario, los cajones negativos no existen").;
```

En algunos casos, este sistema puede ser conveniente, pero es extremadamente raro. Antes de cerrar este apartado, hay que entender un punto clave: un switch permite ejecutar una acción no solo en base a un valor, sino también del tipo del valor (como el operador ===), lo que significa que el código del ejemplo siguiente nunca permitirá que aparezca "¡Bravo!”.

Código: JavaScript

```
var cajon = prompt ('Ingrese el valor 1');

switch (cajon) {
case 1:
alert ('¡Bravo!');
break;

default:

alert ('Perdido');
}
```

De hecho, hemos eliminado la función parseInt () de nuestro código, lo que significa que pasamos una cadena de caracteres a nuestro switch. Dado que también verifica los tipos de valores, el mensaje "¡Bravo ! " nunca aparecerá en la pantalla.

Sin embargo, si cambiamos la primera case para comprobar una cadena en lugar de un número, no se tiene ningún problema:

Código: JavaScript

```
var cajon = prompt ('Ingrese el valor 1');
switch (cajón) {
case '1 ':
alert ('¡Bravo!');
break;

default:
alert ('Lost');
}
```

Condiciones ternarias

Y finalmente aquí está el último tipo de condicional, el ternario. Verás que son muy especiales, en primer lugar, porque son muy rápidas para escribir (pero no leer) y sobre todo porque devuelven un valor.

Para que puedas entender qué escenario se puede utilizar las condiciones ternarias, vamos a empezar por un pequeño ejemplo con la condición if else:

Código: JavaScript

```
Mensajeinicio var = 'Su categoría:',
Mensajefin,
adulto = confirm ('¿Eres mayor de edad? ');
if (adulto) {/ / La variable "adulto" contiene un valor lógico (booleano), podemos por lo tanto, presentar directamente la estructura if sin ningún operador condicional
Mensagefin = '18 + ';
else {}
Mensagefin = '-18';
}
alert (Mensageinicio + Mensagefin);
```

Como se puede ver, el código es bastante largo suficiente para un resultado menor. Con ternarios se puede ayudar a simplificar el código de manera significativa:

Código: JavaScript

```
var Mensageinicio = 'Su categoría:',
Mensagefin,
adulto = confirm ("¿Eres mayor de edad? ');
```

```
Mensagefin = adulto ? '18 + ':' -18 ';
alert (Mensageinicio + Mensagefin);
```

Entonces, ¿cómo es el ternario? Para entender esto debemos mirar la línea 4 del código anterior: Mensagefin = adulto? '18 + ':' -18 ';

Si desglosamos esta línea se puede ver:

- Variable Mensagefin que va a recibir el resultado del ternario;
- Adulto variable que será analizada por el ternario;
- Un signo de interrogación seguido de un valor (número, texto, etc.)
- Dos puntos (:) seguidos de un segundo valor, y, finalmente, punto y coma marca el final de la línea de comandos.

El funcionamiento es sencillo: si la variable adulto es true (verdadera) entonces el valor devuelto por la condición ternaria será el escrito después del signo de interrogación, si es false (falsa), entonces el valor devuelto será el que aparece después de los dos puntos.

No es muy complicado. Las condiciones ternarias son muy rápidas y fáciles de escribir, pero tienen una mala reputación de ser bastante ilegibles (no los note fácilmente en líneas de código). Muchas personas desaconsejan el uso, por nuestra parte te recomendamos que las utilices porque son muy útiles.

Condiciones sobre las variables

JavaScript es un lenguaje único en su sintaxis, te darás cuenta más adelante, si ya sabes otro lenguaje de programación más "convencional". El caso particular que hemos estudiado se refiere a las variables de prueba: es posible probar si una variable tiene un valor sin utilizar la instrucción typeof!.

Prueba de la existencia de contenido de una variable
Para probar la existencia de contenido de una variable, primero debes saber que todo está en la conversión de tipos. Sabrás que las variables pueden ser de varios tipos: números, cadenas de caracteres, etc. Descubrirás que el tipo de una variable (cualquiera que sea) se puede convertir en booleano incluso si tiene una base de número o cadena de caracteres. Un ejemplo simple:

Código: JavaScript

```
var Testcondicion = '¿Va a funcionar? ¿No funcionará ?';
if (Testcondicion) {
alert ('Funciona');
} else {
alert ('No funciona');
```

```
}
```

El código muestra el texto "Funciona". ¿Por qué? Simplemente porque la variable Testcondicion
se convierte en booleana y su contenido se evalúa como true (verdadero). ¿Qué es un contenido verdadero o falso? Simplemente con enumerar los contenidos falso basta para saberlo: un número que es cero o una cadena nula. Es decir, estos dos casos son los únicos que deben ser evaluados como falso. Después de esto es posible evaluar atributos, métodos, objetos, etc. Por supuesto, el valor undefined se evalúa como falso.

Operador O

Su función principal es ser utilizado para devolver la variable primera con un valor evaluado true. Por ejemplo:

Código: JavaScript

```
var  condicionTest1 ='', condicionTest2 = 'Una cadena de caracteres';
alert (conditionTest1 | | conditionTest2);
```

Al final el código devuelve "Una cadena de caracteres". ¿Por qué? Pues porque el operador O se carga para devolver el valor de la primera variable cuyo contenido se evalúa como true. Esto es extremadamente útil.

Resumen

- Una condición devuelve un valor booleano: true o false.
- Existen muchos operadores para verificar condiciones de ensayo y se pueden combinar juntos.
- La condición if else es la más común y permite combinar las condiciones.
- Cuando se trata de probar la igualdad entre varios valores, la condición switch es preferible.
- El ternario es una forma concisa de escribir las condiciones if else y tienen la ventaja de devolver un valor.

Bucles

En este capítulo se hará hincapié en un comportamiento interesante: en realidad se ve cómo se programan las acciones de repetición, para no escribir repetidamente las mismas instrucciones. Pero antes de eso, vamos a discutir el tema del incremento.

Incremento

Considera el cálculo siguiente:

Código: JavaScript

```
var numero = 0;
numero = numero + 1;
```

La variable numero contiene el valor 1. La instrucción para añadir uno es bastante pesada para escribir y recordar, somos perezosos. Javascript, al igual que otros lenguajes de programación, permite lo que se llama incremento y su opuesto decremento,

- Funcionamiento

El incremento agrega una unidad a un número utilizando una sintaxis corta. En contraste, el decremento permite restar uno.

Código: JavaScript

```
var numero = 0;
numero ++;
alert (numero) / / Muestra: "1"
numero --;
alert (numero) / / Muestra: "0"
```

Se trata por lo tanto un método lo suficientemente rápido como para sumar o restar una unidad a una variable (se dice incremento o decremento), y esto será especialmente útil en este capítulo.

- El orden de los operadores

Hay dos formas de uso del incremento, de acuerdo con la posición del operador ++ (o --). Vimos que se podía colocar después de la variable, pero puede situarse delante. Pequeño ejemplo:
Código: JavaScript

```
numero_1 var = 0;
numero_2 var = 0;
número_1++;
++numero_2;
alert (numero_1) / / Muestra: "1"
alert (numero_2) / / Muestra: "1"
```

numero_1 y numero_2, tienen dos incrementos. ¿Cuál es la diferencia entre los dos procedimientos? La diferencia está en el hecho de prioridad de la operación, y es importante si se desea recuperar el resultado del incremento. En el siguiente ejemplo, ++numero devuelve el valor de número incrementado, es decir 1.

Código: JavaScript

```
var numero = 0;
var salida = ++numero;
alert (numero) / / Muestra: "1"
alert (salida) / / Muestra: "1"
```

Ahora, si el operador se coloca después del incremento variable, la operación devuelve el valor del número antes de que se incremente:

Código: JavaScript

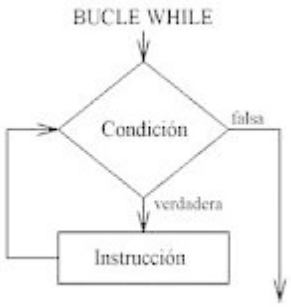
```
var numero = 0;
var salida = número++;
alert (número) / / Muestra: "1"
alert (salida) / / Muestra: "0"
```

Aquí, por lo tanto, la operación numero++ no ha devuelto número con el valor incrementado.

Bucle while

Un bucle es una estructura condicional, similar a lo visto previamente, excepto que se trata de repetir una serie de instrucciones. La repetición es hasta que lo indique el bucle.

Cada vez que el bucle se repite se efectúa una iteración (que en realidad es un sinónimo de repetición).



Funcionamiento del bucle while

Para ejecutar un bucle, es necesario definir una condición. Mientras que sea verdadera (true), el bucle se repite. Una vez que la condición es falsa (false), el bucle se detiene. Aquí se muestra un ejemplo de la sintaxis de un bucle while:

Código: JavaScript

```
while (condicion) {
sentencia_1;
sentencia_2;
sentencia_3;
}
```

Repetir mientras que …

El bucle se repite mientras que la condición sea válida. Esto significa que hay que fijar a la vez, de modo que la condición ya no lo es, de lo contrario el bucle se repetirá indefinidamente. Como ejemplo, vamos a incrementar un número, que es 1, hasta 10:

Código: JavaScript

```
var numero = 1;
while (numero <10) {
número++;
}
alert (numero) / / Muestra: "10"
```

En primer lugar, el número es 1. Al llegar al bucle preguntará si el número es menor que 10. Como es verdadero, se ejecuta el bucle, y el número se incrementa. Siempre que las instrucciones en el bucle se estén ejecutado, la condición del bucle es reevaluada sobre la conveniencia de volver a ejecutar el bucle o no. En este ejemplo, el bucle se repite hasta que el número es igual a 10. Si el número es 10, el número de condición <10 es falso, y se detiene el bucle. Cuando el bucle termina, las instrucciones posteriores al bucle (alert () en nuestro

ejemplo) se ejecutan normalmente.

Ejemplo práctico. Imagina un pequeño script que te preguntará el nombre del usuario y los nombres de sus hermanos y hermanas. No es complicado hacer, se trata de mostrar un cuadro de diálogo con prompt () para cada nombre.

Pero, ¿cómo saber con antelación el número de hermanos y hermanas? Vamos a utilizar un bucle while que pide a cada paso del bucle, un nombre. El bucle sólo se detiene cuando el usuario elige no entrar un nombre.

Código: JavaScript

```
var = nicks = ' ', nick,
        proceso = true;
while (proceso) {
        nick = prompt ('Introduzca un nombre:');

if (nick) {
        nicks += nick + ' ';  / / Agrega el nuevo nombre y un espacio
        después
          } else  {
                proceso = false / / No ha sido inscrito, por lo que hace
                invalidar la condición
        }
}

alert (nicks) / / Muestra los nombres
```

La variables proceso es la que se llama variable de control o variable de bucle. Esta es una variable que no participa directamente en la instrucción del bucle sólo sirve para verificar la condición. Cada iteración del bucle, solicita un nombre y se guarda temporalmente en la variable nick. A continuación, realiza una nick prueba para ver si contiene algo, y en este caso, se suman los nicks prénomà variables. Recuerda que se añade un espacio para separar los nombres. Si, por el contrario nick contiene null - lo que significa que el usuario no ha introducido un nombre o ha pulsado en cancelar, se cambia el valor de proceso a false, que invalida la condición, y evitará que la iteración de bucle se repita.

Algunas mejoras

Uso de break

En los nombres del ejemplo dado, se utiliza una variable de bucle con el fin de detener el bucle. Sin embargo, hay una palabra clave para detener el ciclo de una vez. Esta palabra clave es break y se utiliza exactamente como en la estructura condicional switch, vista en el capítulo

anterior. Si tomamos un ejemplo, esto es lo que sucede con un break::

Código: JavaScript

```
var = nicks = ' ', nick;
while (true) {
        nick = prompt ('Introduzca un nombre:');
        if (nick) {
                nicks += nick + ' ';  / / Agrega el nuevo nombre y un
                espacio después
         } else  {
                break; / / Salir del bucle
        }
}
alert (nicks) / / Muestra los nombres
```

Uso de continue. Esta declaración es más rara, porque las oportunidades de uso no son habituales, continue es algo parecida a break, puede poner fin a una iteración, pero atención, no provoca el fin del bucle, se detiene la iteración actual y pasa a la siguiente.

Do while

Este bucle se asemeja mucho al bucle while, excepto que en este caso el bucle se ejecuta siempre por lo menos una vez. En el caso de un bucle while, si la condición no es válida, el bucle no es ejecutado Con do while, el bucle se ejecuta una vez, entonces la condición se comprueba para determinar si el bucle debe continuar. La sintaxis de un bucle do while es:

Código: JavaScript

```
do {
sentencia_1;
sentencia_2;
sentencia_3;
} while (condición);
```

Hay una diferencia fundamental de escritura respecto al bucle while, que puede hacer ver la diferencia entre ambos. Sin embargo, el uso de do while no es muy común, y es muy posible que no tengas nunca que usarlo porque los programadores suelen emplear un bucle while normal con una condición que hace que siempre se ejecute una vez. Atención a la sintaxis del ciclo do while: hay un punto y coma después del paréntesis de cierre de while.

Bucle for

El bucle for se ve en su aplicación como parecido al bucle while, pero su arquitectura parece complicado al principio. El bucle for es un bucle que funciona de forma bastante simple, pero muy compleja para los principiantes debido a su sintaxis. Una vez que el bucle está controlado, es muy probable que se utilice muy a menudo. El diagrama de un bucle for es:

Código: JavaScript

```
for (inicio; condicion, incremento) {
sentencia_1;
sentencia_2;
sentencia_3;
}
```

En los paréntesis del bucle ya no se encuentra la condición, sino tres bloques: inicio, condición e incremento. Estos tres bloques están separados por un punto y coma, que es un poco como si los paréntesis contuvieran tres instrucciones distintas.

El bucle por lo tanto tiene tres bloques que lo definen. El tercer bloque es el incremento que se utiliza para el incremento de una variable en cada iteración del bucle. Por lo tanto, el bucle es útil para contar y para repetir el bucle un número determinado de veces. En el siguiente ejemplo, vamos a mostrar cinco veces un cuadro de diálogo con alert (), que muestra el número de cada
iteración:

Código: JavaScript

```
for (var iter = 0; iter <5; iter++) {
alert ('Nº. Iteración' + iter);
}
```

En el primer bloque de inicialización, inicializamos una variable llamada iter que vale 0, la palabra clave var se requiere, como cualquier inicialización. Se define en la condición de que el bucle continúa mientras que iter es estrictamente menor que 5. Por último, en el bloque de incremento, se indica que iter se incrementará en cada iteración completa.

Pero me muestra "Iteración n º 4" al final, ¿no hay ninguna iteración 5? Es bastante normal por dos razones: la primera que el bucle parte de 0, así que si contamos desde 0 a 4, hay 5
Velocidad: 0, 1, 2, 3 y 4. La otra, el incremento no tiene lugar antes de cada iteración, sino al final de cada iteración. Por lo tanto, la primera vuelta se hace con iter que es 0, antes de ser incrementado.

Volviendo a nuestro ejemplo

Con los puntos de la teoría que acabamos de ver, podemos volver a escribir nuestro ejemplo de los nombres empleando un bucle for, evitando contar en cada etapa:

Código: JavaScript

```
for (var nicks = ' ', nick; true;) {
	nick = prompt ('Introduzca un nombre:');

		if (nick) {
			nicks += nick + ' ';

		} else {
			break;
		}
	}

	alert (nicks);
```

En el bloque de inicialización (en primer lugar), se comienza inicializando nuestras dos variables. A continuación viene el bloque con la condición (la segunda), que es simplemente true. Se termina con el bloque de incremento .. y no es necesario más aquí, ya que no hay necesidad de incrementar. El tercer bloque está vacío, pero existe. Es por esto por lo que todavía debe poner el punto y coma después del segundo bloque (condición)

Ahora, se va a modificar el bucle para contar cuántos nombres han sido registrados. Para ello, vamos a crear una variable de bucle llamada i, que se incrementa a cada paso del bucle.

Código: JavaScript

```
for (var i = 0, nicks = ' ', nick; true; i++) {
	nick = prompt ('Introduzca un nombre:');
	if (nick) {
		nick nick + + =' ';
	} else {
		break;
	}
}
alert('Hay ' + i + ' nombres :\n\n' + nicks);
```

La variable de bucle se ha añadido en el bloque de inicio. El bloque de incremento también se ha modificado: se indica que hay que incrementar la variable de bucle i. Así, en cada iteración del bucle, se incrementa i, lo que nos permitirá contar el número de nombres agregados.

Alcance de las variables del bucle

En Javascript, no se deben declarar las variables dentro de un bucle (entre llaves), en aras de las prestaciones (velocidad) y la lógica: en efecto, hay no hay necesidad de declarar la misma variable cada vez en el bucle. Es aconsejable declarar las variables directamente en el bloque de inicialización, como se muestra en los ejemplos Pero cuidado: Una vez que se ejecuta el bucle, la variable sigue existiendo, lo que explica que el ejemplo anterior se pueda recuperar el valor de i después que el bucle se completa. Este comportamiento es diferente del de otros lenguajes, en los que una variable declarada en un bucle es "destruida" después de que el bucle se ejecuta.

Prioridad de ejecución

Los tres bloques que forman el bucle for no se ejecutan al mismo tiempo:

- Inicio: justo antes de que comience el bucle. Es como si las instrucciones fueron escritas justo antes del bucle, un poco como un bucle while;

- Condición: antes de cada iteración del bucle, al igual que la condición de un bucle while;

- Incremento: después de cada ciclo. Esto significa que si ejecutas un break en un bucle for, el paso en el bucle a partir de break no será contabilizado.

El bucle for es muy utilizado en Javascript, y no el bucle while, a diferencia de otros lenguajes de
programación. Como veremos más adelante, el funcionamiento real de JavaScript hace que sea necesario en la mayoría de los casos, como la manipulación tablas y de objetos. Se verá más adelante. También hay una variante del bucle llamada for in que sólo se utiliza en casos específicos.

Resumen

- El incremento es importante dentro de los bucles. Incrementar o decrementar significa la adición o sustracción de una unidad a la variable. El comportamiento de un operador de incremento es diferente si se sitúa antes o después de la variable.
- El bucle while permite repetir una lista de instrucciones mientras la condición sea verdadera.
- El ciclo do while es una variante de while, que que será ejecutado al menos una vez, independientemente de la condición.
- El bucle for es un bucle para repetir una lista de instrucciones un número dado de veces. Esta es una variante muy específica del bucle while.