

Introducción a Javascript

Patricio Martínez Cano

July 27, 2017

Contents

Introducción

Qué es javascript

Es un lenguaje de programación que surgió con el objetivo inicial de programar ciertos comportamientos sobre las páginas web, respondiendo a la interacción del usuario y realizando automatismos sencillos. En ese contexto podríamos decir que es un lenguaje de scripting del lado del cliente. Sin embargo hoy en día javascript es mucho más. Las necesidades de aplicaciones de la web moderna y el html5 ha provocado que el uso de javascript que encontramos hoy haya llegado a niveles de complejidad y prestaciones tan grandes como otros lenguajes de primer nivel. Ya no solo lo encontramos en la web ahora también se hacen incluso programas de ordenador en este lenguaje.

En el contexto de un sitio web, con javascript puedes hacer todo tipo de acciones e interacción. Antes se utilizaba para validar formularios, mostrar cajas de dialogo y poco más. Hoy es el motor de muchísimas aplicaciones conocidas.

Entender javascript

Para entender bien lo que es javascript y en qué situaciones se utiliza debes conocer los distintos lenguajes que se usan en la web.

Javascript básico

A javascript se le denomina "del lado del cliente" porque se ejecuta en el navegador (cliente web), en contraposición a otros lenguajes, como PHP que se ejecuta en el servidor. Gracias a su compatibilidad con todos los navegadores modernos se ha convertido en un estándar como lenguaje de programación del lado del cliente.

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 interacciones. Además gracias a las **api javascript de html5**, que están disponibles en los navegadores actuales, podemos acceder a todo tipo de recursos adicionales, como la cámara, espacio para almacenamiento de datos, creación de gráficos, etc.

Cómo y cuando aprender javascript

El mejor momento para aprender javascript es cuando se conozca y domine bien los lenguajes básicos de la web como son el HTML y el css. Es un lenguaje fácil de aprender incluso para aquellos que no tengan experiencia en programación.

Cuando empiezas a aprender javascript, ejecutando tus programas en el navegador, los primeros ejemplos que realizarás tendrán dos vertientes. Por un lado, los **efectos especiales** sobre páginas web, para crear contenidos dinámicos y elementos de la página que tengan movimiento o dinamismo. Por otro lado, nos permite ejecutar instrucciones como respuesta a las acciones del usuario (eventos), con lo que podemos crear **páginas interactivas** con programas como calculadoras, agendas, hojas de cálculo, interfaces de usuario, etc.

Con javascript el programador es capaz de alterar cualquier cosa que se muestre en una página web, cambiando, insertando o eliminando todo tipo de contenido. Si se desea, se puede controlar cualquier cosa que ocurre en la página cuando ésta está siendo visualizada y se puede interaccionar con el usuario.

Librerías de javascript

Una vez conocido el javascript básico se puede pasar a aprender a manejar las muchas librerías que hay desarrolladas y que nos permitirán hacer cosas muy interesantes en mucho menos tiempo.

JQuery

Jquery es la librería más conocida de javascript y se ha convertido en un complemento en la mayoría de las webs que usamos hoy en día, por su facilidad de uso y por su potencia. Con JQuery puedes escribir código javascript que es capaz de ejecutarse sin errores en cualquier navegador e implementa funcionalidades que puedes requerir repetidamente en muchas páginas web.

JQuery te permite además programar nuevas funcionalidades por medio de plugins para hacer cosas tan variadas como validación de formularios, sistemas de plantillas, interfaces avanzadas de usuario, etc.

Otras librerías javascript

A día de hoy existen muchas librerías que nos permiten hacer cosas similares a JQuery o cosas más diversas. Veremos las que podamos y nos dé tiempo.

Javascript y HTML5

La revolución de javascript ha llegado con la incorporación del HTML5. HTML5 lleva gran cantidad de apis para trabajar ya no solo con el navegador, sino con periféricos o elementos del dispositivo, como cámara,

pantalla, espacio de almacenamiento, GPS, etc.

HTML5 ha llegado para estandarizar aún más javascript y crear una serie de especificaciones que siguen todos los fabricantes de navegadores. HTML5 sirve por tanto, para gran cantidad de dispositivos, desde ordenadores, móviles, Smarth Tv, etc.

APIS de HTML5

Si queremos aprovechar html5 tendremos que aprender a manejar una serie de características nuevas de javascript, con una serie de API que nos sirve para trabajar con los más diversos recursos del navegador y del ordenador/dispositivo del usuario. Las API del HTML5 nos permiten extender todavía más las posibilidades de javascript, llegando a situarlo en condiciones similares a los de otros lenguajes de programación.

Llegado a este punto es inevitable hablar del concepto de "Webapp", que son aplicaciones para móviles y tablets que están basadas en HTML5 (HTML + CSS + javascript) y que pueden controlar el dispositivo, por medio de las API, de igual modo que los lenguajes de programación nativos.

HTML5 y compatibilidad

El problema de HTML5 es que no todos los navegadores implementan todas las características del estándar, por ello para usar HTML5 hay que dar una serie de pasos adicionales.

Existen diversas técnicas para aplicar compatibilidad con navegadores antiguos. Una de ellos es usar la librería **Modernizr** que veremos más adelante.

Web components

Después de hablar de HTML5 tenemos que detenernos en explicar otro de los estándares con los que contamos en los navegadores, los **Web Components**. Esta tecnología sirve para extender el HTML creando nuevos componentes que son como si fueran etiquetas nuevas en el lenguaje, que permiten hacer cosas para las que HTML no está preparado. Componentes puedes encontrar de todo tipo, par implementar las más variadas necesidades y además como desarrollador puedes crear tus propios componentes. Ejemplos:

- Web Components
- Polymer

MVC en Javascript

Siguiendo con la secuencia lógica de aprendizaje de javascript, llegamos al MVC. MVC son las siglas del Modelo, Vista y Controlador y se trata de un paradigma de programación que se usa en lenguajes donde se tiene que trabajar con interfaces gráficas, como es el caso de la Web. Propone la separación del código de las aplicaciones por responsabilidades. Los modelos se encargan de trabajar con los datos de la aplicación, las vistas con la presentación y los controladores hacen de conexión entre vistas y modelos. MVC no es algo

específico de javascript, sino que lo encontramos en lenguajes del lado del servidor como PHP o incluso en lenguajes de propósito general como es Java.

En javascript existen varias librerías que podríamos catalogar en este apartado del MVC, pero que realmente no tienen por qué ser MVC puro. A veces nos referimos a ellas como MV*, porque cada una adapta el patrón con diversas aproximaciones y diversas "capas". Realmente sea o no MVC puro, lo importante es que estas librerías son capaces de separar el código por responsabilidades, lógica de presentación, de negocio, etc. Esa separación de responsabilidades es fundamental para crear código fácil de entender, fácil de escribir y sobre todo, de fácil mantenimiento y gran escalabilidad.

Trabajar con paradigmas como MVC es fundamental en el mundo de las aplicaciones web, porque nos permite organizar mejor nuestro código, facilitando el mantenimiento de las aplicaciones. Sin esa organización es habitual que los desarrollos tiendan al caos cuando son muy complejos de realizar y se aumentan considerablemente los costos de mantenimiento. Es por ello que el MVC en Javascript, aunque ha tardado algo más en establecerse que en otros lenguajes, ha llegado para quedarse. O bien los mencionados MV*, donde la figura del controlador se ha llevado a diversas interpretaciones e implementaciones. A veces encontramos sistemas MVR, MVVM... en realidad es todo lo mismo con distintos matices.

Existen diversas librerías para realizar MVC en javascript, entre las más populares tenemos:

- BackboneJS
- EmberJS
- AngularJS
- Angular2
- React
- etc

NodeJS

No podemos dejar de hablar de javascript sin mencionar NodeJS. Se trata de un lenguaje de propósito general pero que tiene como particularidad usar el motor de javascript V8 (el motor de javascript implementado por Google Chrome) para la ejecución de programas. Que sea de propósito general indica que puedes realizar cualquier tipo de aplicación, por lo tanto NodeJS no es un lenguaje web propiamente dicho. Es capaz de servir para el desarrollo web, igual que es también capaz de servidor para muchas otras cosas que no tienen nada que ver. Por ese motivo, aunque a veces se denomina a NodeJS como el javascript del lado del servidor, la realidad es que es mucho más que eso.

A pesar de que sirven para muchas cosas, NodeJS se ha convertido en un lenguaje muy usado por los desarrolladores web. Muchas herramientas del día a día de los desarrolladores están programadas con NodeJS como los gestores de paquetes "npm" o "bower". También muchas herramientas para desarrollo frontend están programadas en NodeJS. Además existen diversos frameworks como SailsJS o ExpressJS que se pueden usar para aplicar el NodeJS en el desarrollo web.

Aprender Javascript te abrirá muchas puertas profesionales

Javascript es el lenguaje que en estos momentos tiene mayor potencial de crecimiento. Como hemos dicho, javascript te sirve para hacer web, pero también para trabajo con dispositivos por medio de las API de HTML5 para crear webapps.

Existe una gran demanda de profesionales con conocimientos avanzados en javascript, necesarios para acometer cualquier tipo de proyecto en la web de última generación, así que todo el tiempo que inviertas en aprender este lenguaje te resultará de mucha utilidad.

Javascript básico

Qué es Javascript

Javascript es un lenguaje de programación utilizado para crear pequeños programas encargados de realizar acciones dentro del ámbito de un 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 de javascript y ejecutarlas para realizar estos efectos e interactividades, de modo que el mayor recurso, y tal vez el único, con el 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.

Algo de 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.

El Web es un sistema de 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 el 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 y estilo y poco más.

Al complicarse los sitios web, una de las primeras necesidades fue que las páginas respondiesen a lagunas 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 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, que por aquel entonces era 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.

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 llamó LiveScript, pero no duró mucho ese nombre, pues antes de lanzar la primera versión del producto se forjó una alianza con Sun Microsystems, creador de Java, para desarrollar en conjunto ese nuevo lenguaje.

La alianza hizo que javascript se diseñara como un hermano pequeño de Java, solamente útil dentro de las páginas web y mucho más fácil de utilizar, de modo que cualquier persona, sin conocimientos de programación pudiese adentrarse en el lenguaje y utilizarlo a sus anchas. Además, para programar javascript no es necesario un kit de desarrollo, ni compilar scripts, ni realizar 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 los distintos navegadores

Como hemos dicho el javascript de Netscape y el de Microsoft Internet Explorer tenía 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 la 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.

Efectos rápidos con Javascript

Para que veamos un poco de forma somera lo que se puede llegar a hacer con javascript vamos a ver tres ejemplos:

Javascript puede cambiar el contenido HTML. Para ello usaremos el método **getElementById**

```

<!DOCTYPE html>
<html>
<body>

<h2>Qué podemos hacer con Javascript</h2>

<p id="demo">Javascript puede cambiar el contenido HTML</p>

<button type="button" onclick='document.getElementById("demo").innerHTML = "Hola Mundo!"'>Pulsame</button>

</body>
</html>

```

Javascript puede cambiar estilos HTML. (CSS)
Ejemplo:

```

<!DOCTYPE html>
<html>
<body>

<h2>Qué podemos hacer con Javascript</h2>

<p id="demo">Javascript puede cambiar el estilo de un elemento HTML.</p>

<button type="button" onclick="document.getElementById('demo').style.fontSize='35px'">Pulsame</button>

</body>
</html>

```

Javascript puede esconder o mostrar elementos HTML
Veamos estos dos ejemplos:
Ejemplo 1:

```

<!DOCTYPE html>
<html>
<body>

<h2>Qué podemos hacer con Javascript</h2>

<p id="demo">JavaScript puede esconder elementos HTML.</p>

```

```
<button type="button" onclick="document.getElementById('demo').style.display='none'">Púlsame</button>

</body>
</html>
```

Ejemplo 2:

```
<!DOCTYPE html>
<html>
<body>

<h2>Qué podemos hacer con Javascript</h2>

<p id="demo" style="display:none">Hola mundo</p>

<button type="button" onclick="document.getElementById('demo').style.display='block'">Púlsame</button>

</body>
</html>
```

Donde va el lenguaje javascript

Javascript se inserta en el documento HTML

Una forma es insertar el código dentro del documento HTML. Esto quiere decir que tenemos que ver la forma de mezclar ambos lenguajes y que puedan convivir sin problemas entre ellos. Para ello se han incluido 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 entre las etiquetas **<SCRIPTS>** distintas. la colocación de estos scripts no es indiferente. Esto lo veremos más adelante.

Formas de ejecutar un scripts javascript en una página

- Ejecución directa

Es el método 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.

- Respuesta a un evento

Es la otra manera de ejecutar un 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.

JavaScript en <head> o en <body>

Podemos poner cualquier número de scripts en un documento HTML.

Los scripts pueden ser localizados en el <body> o en la sección <head> o en ambos.

- JavaScript en <head>

En este ejemplo, una función Javascript es puesta en la sección <head> de una página HTML.

La función es llamada cuando un botón es pulsado.

Ejemplo:

```
<!DOCTYPE html>
<html>

<head>
<script>
function myFunction(){
    document.getElementById("demo").innerHTML= "Párrafo cambiado";
}
</script>
</head>

<body>

<h1>Página web</h1>
<p id="demo">Un párrafo</p>
<button type="button" onclick="myFunction()">Púlsame</button>

</body>
</html>
```

- Javascript en <body>

En este ejemplo, una función JavaScript es localizada en la sección <body> de una página HTML

Esta función es invocada (llamada) cuando un botón es pulsado:

Ejemplo

```
<!DOCTYPE html>
<html>
<body>

<h1>Una página web</h1>
<p id="demo">Un párrafo</p>
<button type="button" onclick="myFunction()">Púlsame</button>

<script>
function myFunction(){
document.getElementById("demo").innerHTML = "Párrafo cambiado";
}
</body>
</html>
```

– El poner el script al final de la sección body se hace para acelerar la carga de la página ya que la compilación del script hace que la visualización de la página sea más lento.

- JavaScript externo

Los Scripts también pueden estar localizados en un fichero externo.

Por ejemplo creamos un fichero con extensión .js

```
function myFunction() {
document.getElementById("demo").innerHTML = "Párrafo cambiado";
}
```

Ahora para usar este script externo lo ponemos como fuente en el atributo **src** de la etiqueta <script>

Ejemplo:

```
<!DOCTYPE html>
<html>
<body>

<script src="miScript.js"></script>

</body>
</html>
```

El script se puede poner en el <body> o en el <head>

- Ventajas de usar un JavaScript externo

Poner los scripts en ficheros externos tiene una serie de ventajas:

- Separamos el HTML y el código
- Hacemos que el HTML y el JavaScript sea más fácil de leer y mantener
- Si cachemos los ficheros JavaScript la carga de las páginas será más rápida

- Referencias externas

No solo podemos referenciar a un fichero local sino también a un recurso que esté en internet poniendo un URL

Ejemplo:

```
<script src="https://www.w3schools.com/js/myScript.js"></script>
```

Salidas de JavaScript

Javascript puede mostrar los datos de diferentes maneras:

- Escribiendo dentro de un elemento HTML usando **innerHTML**
- Escribiendo dentro del documento HTML usando **document.write()**.
- Escribiendo en una caja de alerta usando **window.alert()**
- Escribiendo en la consola del navegador usando **console.log**

Usando innerHTML

Para acceder a los elementos HTML, JavaScript puede usar el método **document.getElementById(id)**

El atributo **id** define al elemento HTML. La propiedad **innerHTML** define el contenido HTML

Ejemplo

```
<!DOCTYPE html>
<html>
<body>

<h1>Mi primera página web</h1>
<p id="demo"></p>
```

```
<script>
document.getElementById("demo").innerHTML = 5 + 6;
</script>

</body>
</html>
```

Usando document.write()

Para propósito de testeo es conveniente usar **document.write()**

Ejemplo

```
<!DOCTYPE html>
<html>
<body>

<h1>Mi primera página web</h1>
<p>Mi primer párrafo</p>

<script>
document.write(5+6);
</script>

</body>
</html>
```

Si usamos document.write() después de que un documento sea totalmente cargado, se borrará todo el HTML existente.

Ejemplo

```
<!DOCTYPE html>
<html>
<body>

<h1>Mi primera página web</h1>
<p>Mi primer párrafo</p>

<button onclick="document.write(5 + 6)">Púlsame</button>

</body>
</html>
```

Usando window.alert()

También podemos mostrar datos con una caja flotante de alerta

```
<!DOCTYPE html>
<html>
<body>

<h1>Mi primera página web</h1>
<p>Mi primer párrafo</p>

<script>
window.alert(5 + 6);
</script>

</body>
</html>
```

Usando console.log

Lo mismo pero enviando la salida a la consola del navegador

```
<!DOCTYPE html>
<html>
<body>

<h1>Mi primera página web</h1>
<p>Mi primer párrafo</p>

<script>
console.log(5 + 6);
</script>

</body>
</html>
```

Sintaxis en JavaScript

La sintaxis son una serie de reglas que definen como los programas son construidos

Programas Javascript

Un programa de computadora es una lista de "instrucciones" que serán ejecutadas por la computadora.

En un lenguaje de programación, esas instrucciones se llaman **declaraciones**. En javascript esas declaraciones están separadas por punto y coma.

Ejemplo

```
var x, y, z;  
x = 5;  
y = 6;  
z = x + y;
```

Los programas serán ejecutados en el navegador web.

Declaraciones en JavaScript

Las declaraciones en Javascript está compuestas por:

- Valores
- Operadores
- Expresiones
- Palabras clave
- Comentarios

Valores en JavaScript

La sintaxis de javascript define 2 tipos de valores: Valores fijos y valores variables.

Los valores fijos son llamados **literales**. Los valores variables son llamados **variables**

- Valores literales

Los tipos más importantes de valores literales son:

Números que son escritos con o sin decimales

10.50

1001

Veamos un ejemplo:

```

<!DOCTYPE html>
<html>
<body>

<h2>Números en JavaScript</h2>

<p> Los números pueden ser escritos con o sin decimales</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = 10.50;
</script>
</body>
</html>

```

Strings o cadenas de texto. Se escriben con comillas dobles o simples

"Paquito Chocolatero"

'Paquito Chocolatero'

Veámoslo con un ejemplo:

```

<!DOCTYPE html>
<html>
<body>

<h2>Cadenas de texto en JavaScript</h2>

<p>Las cadenas de texto pueden ser escritas con comillas simples o dobles</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = 'Paquito Chocolatero';
</script>
</body>
</html>

```

- Valores variables

En los lenguajes de programación las variables son usadas para **guardar** datos.

En JavaScript usaremos la palabra clave **var** para declarar una variable. El signo igual es lo que usamos para **asignar valores** a las variables.

En este ejemplo, x es definida como una variable. Entonces a x se le asigna el valor 6.

```
var x;
```

```
x = 6;
```

Ejemplo:

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<h2>Variables en JavaScript</h2>
```

```
<p>Esto es un ejemplo, de como definir una variable en JavaScript. Primero tenemos que definir
```

```
<p id="demo"></p>
```

```
<script>
```

```
var x;
```

```
x = 6;
```

```
document.getElementById("demo").innerHTML = x;
```

```
</script>
```

```
</body>
```

```
</html>
```

Operadores en javascript

JavaScript usa **operadores aritméticos** para calcular valores. (+ - * /)

(5 + 6) * 10

Expresiones en JavaScript

Una expresión es una combinación de valores, valores y operadores los cuales calculan un valor. El cálculo es llamado **evaluación**

Por ejemplo 5 * 10

Las expresiones también puede tener variables. x * 10

Los valores pueden tener varios tipos, como números y cadenas de texto.

Un ejemplo es que podemos sumar dos cadenas:

Ejemplo:

```
<!DOCTYPE html>
<html>
<body>

<h2>Variables en JavaScript</h2>
<p>Esto es un ejemplo, de como definir una variable en JavaScript. Primero tenemos que definir la v

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = "Paquito" + " " + "Chocolatero";
</script>

</body>
</html>
```

Palabras claves en JavaScript

Las palabras claves son usadas para identificar acciones que son desempeñadas.

Por ejemplo ya hemos visto la palabra clave **var** para crear variables.

Comentarios en JavaScript

No todo es ejecutado en JavaScript.

El código que pueda haber después de dos barras (//) o entre una barra y un asterisco (/*) es tratado como un comentario.

Los comentarios son muy importantes ya que disponen de información para los humanos que la máquina no tiene que ejecutar.

Ejemplo:

```
var x = 5; //Esto es un comentario
```

Identificadores en JavaScript

Los identificadores son nombres.

En javascript, los identificadores son usados para nombrar variables.

Las reglas para crear nombres "legales" son muy importantes en los lenguajes de programación. signo de dolar En JavaScript, el primer carácter debe ser **una letra**, un **guión bajo** (`_`) o un **signo de dolar** (`$`). Los caracteres siguientes pueden ser letras, dígitos, guiones bajos o signos de dolares.

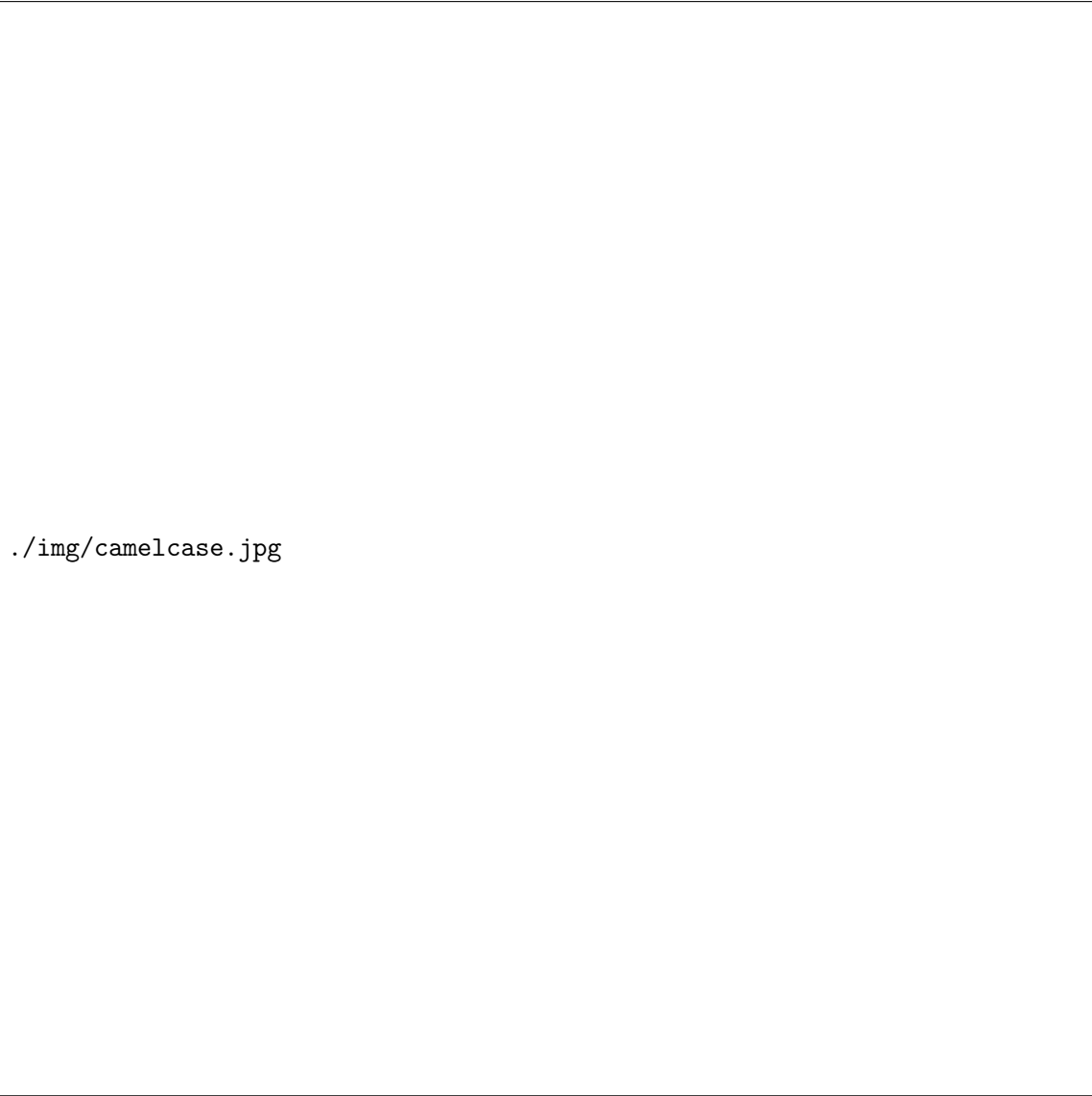
- JavaScript es Case Sensitive

Esto es que todos los identificadores diferencian entre mayúsculas y minúsculas. Por ejemplo no es lo mismo Nombre que nombre.

- JavaScript y el Camel Case

Históricamente, los programadores han usado muchas diferentes formas de unir varias palabras en un nombre de variable. Por ejemplo:

- Palabras separadas entre guiones. Como por ejemplo, primer-nombre, master-card, torre-lodones.
- Guiones bajos. `primer_nombre`, `master_card`, `torre_lodones`.
- CamelCase alto. `PrimerNombre`, `MasterCard`, `TorreLodones`
- Camelcase bajo. Igual que el anterior pero la primera palabra es en minúscula. `primerNombre`, `masterCard`, `torreLodones`. Este es el estilo que usan los programadores de JavaScript



`./img/camelcase.jpg`

Declaraciones en JavaScript

En HTML, las declaraciones de JavaScript son instrucciones que serán ejecutadas por el navegador web.

Declaraciones en JavaScript

Una declaración le dice al navegador que escriba "Hola Mundo" dentro de un elemento HTML con el id demo.

```
document.getElementById("demo").innerHTML = "Hola Mundo";
```

Programas en JavaScript

Un conjunto de declaraciones JavaScript conforman un programa.

Las declaraciones son ejecutadas una por una en el mismo orden en el que están escritas.

Un ejemplo sería:

```
<!DOCTYPE html>
<html>
<body>

  <h2><Declara></>Declaraciones en JavaScript</h2>
  <p>Un Código JavaScript <o></o> simplemente un programa JavaScript es una secuencia de declaraciones.
<p id="demo"></p>

<script>
var x, y, z;
x = 5;
y = 6;
z = x + y;
document.getElementById("demo").innerHTML = z;
</script>

</body>
</html>
```

Puntos y comas

Los puntos y las comas es la forma de separar las declaraciones en JavaScript

Cuando separamos con puntos y comas podemos poner multiples declaraciones en una línea.

Espacios en blanco

JavaScript ignora múltiples espacios. Tu puedes añadir espacios en blanco para hacerlo más legible.

Las siguientes líneas son equivalentes:

```
var persona = "Paquito";
var persona="Persona";
```

Una buena práctica es poner espacios alrededor de los operadores (= + - * /)

```
var x = y + z;
```

Longitud de línea en JavaScript y saltos de línea

Para una mejor lectura de código, los programadores no crean líneas de más de 80 caracteres. Si una declaración no cabe en una sola línea lo mejor es hacer un salto de línea después de un operador.

Ejemplo:

```
document.getElementById("demo").innerHTML =  
    "Hola mundo";
```

Bloques de código

Las declaraciones de JavaScript pueden ser agrupadas en bloques de código, dentro de corchetes { }.

El propósito de un bloque de código es para definir que declaraciones serán ejecutadas. Esto es para funciones.

Ejemplo:

```
function myFunction(){  
    document.getElementById("demo1").innerHTML = "Hola Mundo";  
    document.getElementById("demo2").innerHTML = "Que tal?";  
}
```

Palabras clave en JavaScript

Las declaraciones de JavaScript empiezan con una **palabra clave** que identifica que acción se va a realizar.

Aquí una lista de palabras clave que vamos a aprender en este curso:

| Palabra clave | Descripción |
|---------------|---|
| break | Termina un bucle |
| continue | salta fuera del bucle y empieza de nuevo |
| debugger | Para la ejecución de JavaScript y llama a la función de localización de fallos |
| do ... while | Ejecuta un bloque de declaraciones y repite el bloque mientras una condición sea cierta |
| for | Marca un bloque de declaraciones para ser ejecutado y se hará mientras una condición sea cierta |
| function | Declara una función |
| if ... else | Marca un bloque de declaraciones para ser ejecutado dependiendo de una función |
| return | Sale de una función |
| switch | Marca un bloque de declaraciones para ser ejecutado dependiendo de diferentes casos |
| try ... catch | Implementa un manejador de errores a un bloque de declaraciones |
| var | Declara una variable |

Comentarios en JavaScript

Los comentarios pueden ser usados para explicar el código y hacerlo más legible.

Comentarios de una sola línea

Una línea de comentario empieza con dos barras inclinadas `//`.

Ejemplo:

```
//Ejemplo de comentarios de una línea  
var x = 5; //Declaramos x y le damos el valor de 5  
var y = x + 5; // Declaramos y, y le damos el valor de x + 2
```

Comentarios multi-línea

Un comentario multi-línea empieza con `/*` y termina con `*/`

Ejemplo:

```
/* El código a continuación cambiará la cabecera  
con el id="myH" y el párrafo con el id = "myP"  
en la página web:  
*/  
document.getElementById("myH").innerHTML = "Mi primera página";  
document.getElementById("myP").innerHTML = "Mi primer párrafo";
```

Usando comentarios para prevenir la ejecución

Comentando código para prevenir la ejecución es una forma rápida de hacer testeo de código.

Variables en JavaScript

Variables en JavaScript

Como ya hemos dicho las variables son contenedores de datos.

Ejemplo:

```
var x = 5;  
var y = 6;  
var z = x + y;
```

En este ejemplo podemos ver:

- x guarda el valor 5

- y guarda el valor 6
- z guarda el valor 11

Lo importante es que quede claro que las variables como en álgebra almacenan valores.

Identificadores en Javascript

Todas las variables en javascript deben estar **identificadas** con un **nombre único**

Esos nombres únicos se llaman **identificadores**.

Los identificadores pueden ser nombres cortos o nombres más descriptivos. En general, las reglas para construir identificadores son:

- Los nombres pueden contener letras, dígitos, guiones bajos y signos de dolar
- Los nombres deben empezar con una letra
- Los nombres también puede empezar con un guión bajo y un signo de dolar (aunque no usaremos esta forma)
- Los nombres distinguen entre mayúsculas y minúsculas
- Las palabras reservadas no pueden ser usadas

Declarando variables en JavaScript

Crear una variable se llama **declarar** una variable.

Para declarar una variable usamos la palabra **var**

```
var carName;
```

Ahora la variable está **indefinido**, así que para asignarle un valor usamos el signo igual:

```
carName = "Volvo";
```

También podemos asignar un valor al mismo tiempo que declaramos la variable.

```
var carName = "Volvo";
```

Ejemplo:

```

<!DOCTYPE html>
<html>
<body>
<h2>Variables en JavaScript</h2>

<p>Declara una variable, asigne un valor y muéstrala</p>

<p id="demo"></p>

<script>
var carName = "Volvo";
document.getElementById("demo").innerHTML = carName;
</script>

</body>
</html>

```

También podemos definir muchas variables en una sola línea usando **comas**.

```
var person = "Paquito Chocolatero", carName = "Volvo", precio = 200;
```

Tipos de datos en JavaScript

Las variables en javascript pueden ser cadenas de texto o números con o sin decimales. El asunto es que en JavaScript a diferencia de otros lenguajes soporta los tipos dinámicos, esto es, que no es necesario definir el tipo de la variable sino que el compilador sabrá que tipo de valor es al verlo.

Re-declarando variables

Si tú re-declaras una variable en JavaScript ésta no perderá su valor.

```

var carName = "Volvo";
var carName;

<!DOCTYPE html>
<html>
<body>
<h2>Variables en JavaScript</h2>

<p>Si re-declaremos una variable uno perderemos su valor</p>

```



```

<p id="demo"></p>

<script>
var carName = "Volvo";
var carName;
document.getElementById("demo").innerHTML = carName;
</script>

</body>
</html>

```

Operadores en JavaScript

Operadores aritméticos

Por un lado tenemos los operadores aritméticos:

| Operador | Descripción |
|----------|----------------|
| + | Suma |
| - | Resta |
| * | Multiplicación |
| % | Resto |
| ++ | Incremento |
| -- | Decremento |

Describiremos estos operadores más adelante

Operadores de asignación

Los operadores de asignación asignan valores a las variables

| Operador | Ejemplo | Igual a |
|----------|---------|-----------|
| = | x = y | x = y |
| += | x += y | x = x + y |
| -= | x -= y | x = x - y |
| *= | x *= y | x = x * y |
| /= | x /= y | x = x / y |
| %= | x %= y | x = x % y |

Ejemplo:

```

<!DOCTYPE html>
<html>
<body>
<h2>Operadores en JavaScript</h2>

<p id="demo"></p>

<script>
var x = 10;
x += 5;
document.getElementById("demo").innerHTML = x;
</script>
</body>
</html>

```

Veremos esto con más detenimiento en un futuro

Operadores de cadenas

Como ya hemos visto también podemos usar el operadores para concatenar cadenas de texto.

```

texto1 = "Paquito";
texto2 = "Chocolatero";
texto3 = texto1 + " " + texto2;

```

Esto mismo también podríamos haberlo hecho:

```

texto1 = "Paquito";
texto1 += "Chocolatero";

```

Ejemplo:

```

<!DOCTYPE html>
<html>
<body>
<h2>Operadores en JavaScript</h2>
<p id="demo"></p>

<script>
texto1 = "Paquito";
texto1 += "Chocolatero";

```

```
document.getElementById("demo").innerHTML = texto1;  
</script>  
</body>  
</html>
```

También podemos sumar número y texto:

```
x = 5 + 5;  
y = "5" + 5;  
z = "Hola" + 5;
```

Resultados:

```
10  
55  
hola5
```

Operadores de comparación

| Operador | Descripción |
|----------|--------------------------------------|
| == | igual a |
| = | valor equivalente y tipo igual |
| != | no igual |
| !== | valor no equivalente y tipo no igual |
| > | mayor que |
| < | menor que |
| >= | mayor o igual que |
| <= | menor o igual que |
| ? | operador ternario |

Esto lo veremos más detenidamente en su sección

Operadores lógicos en JavaScript

| Operador | Descripción |
|----------|-------------|
| && | y lógico |
| | or lógico |
| ! | no lógico |

Los operadores lógicos serán mejor explicados en la sección de comparaciones en JavaScript

Operadores de tipos en JavaScript

| Operador | Descripción |
|-------------------------|--|
| <code>typeof</code> | Devuelve el tipo de una variable |
| <code>instanceof</code> | Devuelve verdadero si un objeto es una instancia de un objeto tipo |

Esto lo veremos más adelante en la sección de Conversión de tipos en JavaScript

Operadores aritméticos en JavaScript

| Operador | Descripción |
|-----------------|----------------|
| <code>+</code> | Suma |
| <code>-</code> | Resta |
| <code>*</code> | Multiplicación |
| <code>%</code> | Resto |
| <code>++</code> | Incremento |
| <code>--</code> | Decremento |

Operadores aritméticos

Con estos operadores podemos operar con números:

```
var x = 100 + 50;
```

o variables

```
var x = a + b;
```

o expresiones:

```
var x = (100 + 50) * a;
```

Precedencia de operadores

La precedencia de operadores es lo que nos describe el orden en el que las operaciones serán realizadas en una expresión aritmética.

Ejemplo:

```
var x = 100 + 50 * 3;
```

Vamos a probar:

```
<!DOCTYPE html>
<html>
<body>

<p>Precedencia de operadores</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = 100 + 50 * 3;
</script>

</body>
</html>
```

Como vimos en la escuela la multiplicación y la división van antes que la suma y la resta. También vimos en la escuela que la precedencia puede ser cambiada usando paréntesis

```
var x = (100 + 50) * 3;
```

Cuando en cambio tenemos operadores de la misma precedencia (como por ejemplo la suma y la resta) se calculan de izquierda a derecha.

Ejemplo:

```
var x = 100 + 50 - 3;
```

Asignación en JavaScript

Operadores de asignación en JavaScript

Los operadores de asignación asignan valores a las variables

| Operadores | Ejemplo | Igual que |
|------------|---------|------------|
| = | x = y | x = y |
| += | x += y | x = x + y |
| -= | x -= y | x = x - y |
| *= | x *= y | x = x * y |
| /= | x /= y | x = x / y |
| %= | x %= y | x = x % y |
| «= | x «= y | x = x « y |
| »= | x »= y | x = x » y |
| »>= | x »>= y | x = x »> y |
| &= | x &= y | x = x & y |
| ^= | x ^= y | x = x ^ y |
| l= | x l= y | x = x l y |
| **= | x **= y | x = x ** y |

Ejemplos:

```
var x = 10;
```

```
var x = 10;
```

```
x += 5;
```

```
var x = 10;
```

```
x -= 5;
```

```
var x = 10;
```

```
x *= 5;
```

```
var x = 10;
```

```
x /= 5;
```

```
var x = 10;
```

```
x %= 5;
```

Ejercicios

Tipos de datos en JavaScript

Tipos de datos en JavaScript

Las variables en javascript pueden tener varios tipos de datos: números, cadenas de texto, objetos y más:

```
var length = 16; // Número
```

```
var nombre = "Paquito"; // Cadena de texto
```

```
var x = {nombre:"Paquito", apellido:"Chocolatero"}; // Objeto
```

El concepto de tipo de datos

En programación los tipos de datos es un concepto importante.

Para ser capaz de operar con variables, es importante saber sobre el tipo de dato con el que estamos trabajando. Sin los tipos de datos una computadora no puede resolver esto:

```
var x = 16 + "Volvo";
```

Qué resultado es el que producirá.

JavaScript tratará esto de la siguiente manera:

```
var x = "16" + "Volvo";
```

Veamos una serie de ejemplos y qué resultados vamos a obtener:

```
var x = 16 + "Volvo";
```

```
var x = "Volvo" + 16;
```

JavaScript evalúa las expresiones de izquierda a derecha. Diferentes secuencias puede producir diferentes resultados:

```
var x = 16 + 4 + "Volvo";
```

Resultado:

20Volvo

```
var x = "Volvo" + 16 + 4;
```

Resultado:

Volvo164

¿Vemos la diferencia?

En el primer ejemplo los números son tratados como tales hasta que llega la palabra Volvo. En el segundo ejemplo, una vez tratada la palabra Volvo como una cadena de texto, javascript lo hace igual con el resto.

Los tipos en JavaScript son dinámicos

Los tipos en JavaScript son dinámicos lo que quiere decir que la misma variable puede ser usada con diferentes tipos:

```
var x;  
var x = 5;  
var x = "Paquito";
```

Cadenas de texto en JavaScript

Para crear una cadena de texto usamos las comillas bien sean simples o dobles:

```
var carName = "Volvo XC60";  
var carName = 'Volvo XC60';
```

Podemos usar comillas dentro de una cadena de texto pero no tiene que ser iguales a las comillas que rodean a la cadena de texto:

```
var nombre = "Paquito Chocolatero";  
var nombre = "Paquito 'el pesado' Chocolatero";  
var nombre = 'Paquito "el pesado" Chocolatero';
```

Vamos a verlo en un ejemplo:

```
<!DOCTYPE html>  
<html>  
<body>  
  
<p id="demo"></p>  
  
<script>  
var carName1 = "Volvo XC60";  
var carName2 = 'Volvo XC60';  
var nombre1 = "Paquito Chocolatero";  
var nombre2 = "Paquito 'el pesado' Chocolatero";  
var nombre3 = 'Paquito "el pesado" Chocolatero';  
  
document.getElementById("demo").innerHTML =  
carName1 + "<br>" +  
carName2 + "<br>" +  
nombre1 + "<br>" +  
nombre2 + "<br>" +  
nombre3;  
</script>  
  
</body>  
</html>
```


Números en JavaScript

En Javascript los números solo tienen un tipo. Los números se pueden escribir con o sin decimales.

```
var x1 = 34.00;
var x2 = 34;
```

También podemos usar notación científica:

```
var y = 123e5; // 12300000
var z = 123e-5; // 0.00123
```

Booleanos en JavaScript

Los booleanos solo pueden tener dos valores: true o false.

```
var x = true;
var y = false;
```

Arrays en JavaScript

Un array (o arreglo) es un conjunto de datos que comparten el mismo nombre pero que se diferencian por un índice. Los arrays o arreglos van entre corchetes y separados por comas.

Por ejemplo, podemos hablar de varias marcas de coche:

```
var cars = ["Ford", "Volvo", "BMW"];
```

El índice empieza por 0 y el siguiente es el 1 y así.

Objetos en JavaScript

Los objetos se escriben entre llaves.

Los objetos son parejas de nombre:valor y están separados por comas.

```
var person = {nombre:"Paquito", apellido:"Chocolatero", edad: 100};
```

El operador typeof

Podemos usar el operador **typeof*** para saber el tipo de variable con el que estamos trabajando.

Ejemplo:

```
typeof "" //devuelve "string"
typeof "Paquito" //devuelve "string"
typeof 0 // devuelve "number"
typeof 3.44 // devuelve "number"
```

Datos primitivos

Un valor primitivo es aquel que es un dato simple sin propiedades adicionales ni métodos. Datos primitivos son:

- cadena de texto
- número
- booleano
- null
- indefinido

Datos complejos

Los datos complejos son dos:

- funciones
- objetos

Indefinidos

En JavaScript, una variable sin un valor tiene el valor **indefinido**

Valores vacíos

Un valor vacío no tiene nada que ver con un indefinido. Un valor vacío tiene valor y tipo.

```
var car = "";    // El valor es "", y typeof devuelve "string"
```

Null

In JavaScript null es "nada". Se supone que es algo que no existe. Por desgracia, en JavaScript, el tipo de dato null es un objeto.

Diferencia entre Indefinido y Null

```
typeof undefined //undefined
typeof null // object
null === // false
null == undefined // true
```

Ejemplo:

```
<!DOCTYPE html>
<html>
<body>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML =
typeof undefined + "<br>" +
typeof null + "<br>" +
(null === undefined) + "<br>" +
(null == undefined);
</script>

</body>
</html>
```

Funciones en JavaScript

Una función es un bloque de código diseñado para ejecutar una determinada tarea. Una función es ejecutada cuando "algo" la invoca.

Ejemplo:

```
function myFunction(p1, p2){
    return p1 * p2;
}
```

Veamos lo en acción:

```
<!DOCTYPE html>
<html>
<body>

<h2>Funciones en Javascript</h2>

<p>En este ejemplo llamamos a una función que hace un cálculo y muestra un resultado</p>

<p id="demo"></p>
```

```

<script>
function myFunction(p1, p2) {
    return p1 * p2;
}
document.getElementById("demo").innerHTML = myFunction(4, 3);
</script>

</body>
</html>

```

Sintaxis de las funciones en JavaScript

Para definir una función usamos la palabra clave **function** seguido del nombre de la función y seguido de unos paréntesis. Los nombres de las funciones pueden contener letras, dígitos, guiones bajos y signos de dólares (lo mismo que con las variables).

Los paréntesis pueden incluir parámetros separados por comas. El código que se ejecuta se encuentra entre las llaves.

Invocación de la función

El código dentro de la función se ejecuta cuando "algo" la invoca:

- Cuando un evento ocurre (un usuario pulsa un botón)
- Cuando es invocado desde el código de JavaScript
- Automáticamente (auto invocación)

La función Return

Cuando en el código aparece la palabra **return**, la función dejará de ejecutarse.

Si la función es invocada desde una declaración, JavaScript volverá a ejecutar el código antes de la invocación de la declaración.

Las funciones a menudo calculan un **valor de retorno**. Éste valor de retorno es devuelto al solicitante.

Ejemplo:

```

var x = myFunction(4, 3); // La función es llamada y devuelve un valor a x

function myFunction(a, b) {
    return a * b;          // La función devuelve el producto de a por b
}

```

Vamos a verlo funcionando:

```
<!DOCTYPE html>
<html>
<body>

<h2>Funciones en JavaScript</h2>

<p>Ejemplo de una llamada a una función y la devolución del valor</p>

<p id="demo"></p>

<script>
function myFunction(a, b) {
    return a * b;
}
document.getElementById("demo").innerHTML = myFunction(4, 3);
</script>

</body>
</html>
```

Por qué funciones

Por un tema de reutilización de código. Una vez creado el código puedes volver a usar esa función tantas veces como quieras.

Ejemplo. Función para convertir de grados Fahrenheit a Celsius

```
function aCelsius(fahrenheit){
    return (5/9) * (fahrenheit-32)
}
```

Vamos a verlo en la web

```
<!DOCTYPE html>
<html>
<body>

<h2>Funciones en JavaScript</h2>

<p>En este ejemplo llamamos a una función que convierte de grados Fahrenheit a Celsius</p>
```

```
<p id="demo"></p>

<script>
function toCelsius(f) {
    return (5/9) * (f-32);
}
document.getElementById("demo").innerHTML = toCelsius(77);
</script>

</body>
</html>
```

El operador () invoca a la función

Usando funciones como valores de variables

Las funciones pueden ser usadas al igual que usamos las variables,
Ejemplo:

```
var x = aCelsius(77);
var texto = "La temperatura es" + x + "grados Celsius";
```

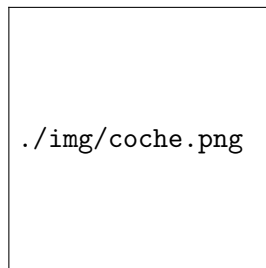
También podemos usar directamente la función como el valor de una variable:

```
var texto = "La temperatura es" + aCelsius(77) + "grados Celsius";
```

Objetos en JavaScript

La programación orientada a objetos intenta simular el mundo real donde tenemos objetos físicos con los que trabajar. Vamos a poner el ejemplo de un coche que es lo más habitual.

Un coche tiene **propiedades** tales como el peso o el color y **métodos** como arrancar y parar.



| Objeto | Propiedades | métodos |
|--------|----------------------|------------------|
| coche | coche.nombre = Fiat | coche.arranar() |
| | coche.modelo = 500 | coche.conducir() |
| | coche.peso = 850kg | coche.frenar() |
| | coche.color = blanco | coche.parar() |

Todos los coches tienen las mismas propiedades, pero el valor de las propiedades difiere de un coche a otro. Además todos los coches tienen los mismos métodos, pero los métodos son desarrollados en diferentes momentos.

Objetos en JavaScript

Como ya hemos visto en JavaScript podemos fácilmente asignar valores a variables:

```
var coche = "Fiat";
```

Los objetos son variables también. Pero los objetos pueden tener muchos valores. Con este código asignamos muchos valores a la variable coche:

```
var car = {type:"Fiat", model:"500", color:"blanco"};
```

Propiedades de los objetos

En el par nombre:valor es llamado **propiedad**

| Propiedad | Valor |
|-----------|-------------|
| Nombre | Paquito |
| Apellidos | Chocolatero |
| edad | 100 |

Métodos de un objetos

Los métodos son acciones que puede desarrollar un objeto. Los métodos son guardados en propiedades como una **definición de función**.

| Propiedad | Valor |
|----------------|---|
| nombre | Paquito |
| apellido | Chocolatero |
| edad | 100 |
| nombreCompleto | function(){return this.nombre + " " + this.apellido;} |

Definición de objeto

Para definir (crear) un objeto en JavaScript lo hacemos de la siguiente manera:

```
var persona = {nombre:"Paquito", apellido:"Chocolatero", edad:100};
```

Los espacios y saltos de línea no son importantes. Una definición de objeto puede hacerse con múltiples saltos de línea:

```
var persona = {  
  nombre:"Paquito",  
  apellido:"Chocolatero",  
  edad:100  
}
```

Accediendo a las propiedades de un objeto

Se puede acceder a las propiedades de un objeto de dos formas:

`nombreObjeto.nombrePropiedad`

o

`nombreObjeto["nombrePropiedad"]`

Ejemplo:

`persona.apellido;`

o

`persona["apellido"];`

Accediendo a los métodos de un objeto

Para acceder a los métodos de un objeto usamos la siguiente sintaxis:

`nombreObjeto.nombreMetodo()`

Ejemplo:

`nombre = persona.nombre();`

Si omitimos los paréntesis, nos devolverá una **definición de la función**, esto es, nos describirá el método. Ejemplo

```
<!DOCTYPE html>
<html>
<body>

<p>Creando un método de objeto</p>

<p>Un método es una definición de función, guardada como un valor de una propiedad.</p>

<p id="demo"></p>

<script>
var persona = {
  nombre: "Paquito",
  apellido: "Chocolatero",
  id: 8899,
  nombreCompleto: function(){
    return this.nombre + " " + this.apellido;
  }
};

document.getElementById("demo").innerHTML = persona.nombreCompleto;
</script>
</body>
</html>
```

No declarar cadenas de texto, números y booleanos como objetos.

Cuando una variable en JavaScript es declarada con la palabra clave **new**, esa variable se crea como un objeto.

```
var x = new String();
var y = new Number();
var z = new Boolean();
```

Esto complica el código y hace que la velocidad de ejecución baje. Veremos esto más adelante.

Ejercicios

Ámbito en JavaScript

El ámbito es desde donde se puede acceder a ellas dentro de nuestro programa.

En JavaScript, objetos y funciones también son variables.

Variables locales en JavaScript

Las variables declaradas dentro de una función locales a esas funciones, es decir no se puede acceder desde fuera de ellas.

Así las variables locales tienen un **ámbito local**.

Ejemplo:

```
// El código de aquí no puede acceder a la variable nombreCoche
function myFunction() {
    var nombreCoche = "Volvo";

    // El código de aquí puede acceder a la variable nombreCoche
}
```

```
<!DOCTYPE html>
<html>
<body>
```

```
<p>La variable local nombreCoche no puede ser accedida desde fuera de la función</p>
```

```
<p id="demo"></p>
```

```
<script>
myFunction();
document.getElementById("demo").innerHTML =
"The type of carName is " + typeof carName;
```

```
function myFunction() {
    var carName = "Volvo";
}
</script>
```

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

Las variables locales solo son reconocidas dentro de sus funciones por lo que se puede usar el mismo nombre de variable en distintas funciones. Las variables locales son creadas cuando la función empieza y se eliminan cuando la función es completada.

Variables Globales en JavaScript

Una variable declarada fuera de una función, se vuelve **global**Null

Una variable global tiene **ámbito global**. Todos los scripts y funciones de una página web puede acceder a ella.

Ejemplo

```
var nombreCoche = "Volvo";

// el código de aquí puede usar nombreCoche

function myFunction(){
    // el código de aquí puede usar nombreCoche
}
```

Ejemplo en web:

```
<!DOCTYPE html>
<html>
<body>

<p>A una variable global se puede acceder desde cualquier función o script de la página web.</p>

<p id="demo"></p>

<script>
var carName = "Volvo";
myFunction();

function myFunction() {
    document.getElementById("demo").innerHTML =
        "Puedo mostrar " + carName;
}
</script>

</body>
</html>
```

Ámbito global de forma automática

Si asignamos un valor a una variable que no ha sido declarada, ésta automáticamente se convertirá en una variable **global**

Ejemplo

```
myFunction();  
  
// el código aquí puede usar nombreCoche  
  
function myFunction(){  
    nombreCoche = "Volvo";  
}
```

Variables globales en HTML

Con JavaScript, el ámbito global es el entorno de JavaScript completo. En HTML, el ámbito global es el objeto ventana. Todas las variables globales pertenecen al objeto ventana.

```
var nombreCoche = "Volvo";  
  
// el código aquí puede usar window.nombreCoche
```

Tiempo de vida de una variable en JavaScript

El tiempo de vida de una variable empieza cuando ésta es declarada. Las variables locales serán borradas cuando la función es completada. En un navegador web, las variables globales son borradas cuando cerramos la ventana del navegador (o pestaña), pero permanece disponible para nuevas páginas en la misma ventana.

Argumentos de función

Los argumentos de una función (parámetros) funcionan como variables locales dentro de las funciones.

Eventos en JavaScript

Los eventos en HTML son "cosas" que ocurren a los elementos HTML. Cuando JavaScript es usado en páginas HTML, puede "reaccionar" a esos eventos.

Eventos HTML

Un evento HTML puede ser algo que el navegador hace o que el usuario hace. Algunos ejemplos de eventos son:

- Una página de HTML es cargada completamente
- Un campo de entrada es cambiado
- Un botón es pinchado

JavaScript permite ejecutar código cuando un evento es detectado. HTML permite a los eventos manejar atributos con código javascript y ser añadido a los elementos HTML. Lo podemos hacer con comillas dobles o simples.

Ejemplos

```
<button onclick="document.getElementById('demo').innerHTML = Date()">La hora es</button>

<!DOCTYPE html>
<html>
<body>

<button onclick="document.getElementById('demo').innerHTML=Date()">The time is?</button>

<p id="demo"></p>

</body>
</html>
```

Ahora vamos a hacer que cambie directamente el propio elemento:

```
<!DOCTYPE html>
<html>
<body>

<button onclick="document.getElementById('demo').innerHTML=Date()">The time is?</button>

<p id="demo"></p>

</body>
</html>
```

Eventos comunes en HTML

| Evento | Descripción |
|-------------|--|
| onchange | Un elemento ha sido cambiado |
| onclick | El usuario ha pulsado un elemento |
| onmouseover | El usuario mueve el ratón sobre un elemento |
| onmouseout | El usuario aleja el ratón del elemento |
| onkeydown | El usuario pulsa una tecla del teclado |
| onload | El navegador ha finalizado la carga de la página |

Qué puede hacer JavaScript

Los eventos pueden ser usados para manejar y verificar la entrada del usuario, sus acciones y las acciones del navegador:

- Cosas que pueden hacerse cuando se carga una página
- Cosas que pueden hacerse cuando se cierra una página
- Acciones que pueden realizarse cuando se pulsa un botón
- Contenido que puede ser verificado cuando un usuario introduce datos
- Y más ...

Muchos diferentes métodos pueden ser usados para permitir a JavaScript funcionar con eventos:

- Atributos de eventos pueden ser cambiados cuando se ejecuta código JavaScript directamente
- Atributos de eventos pueden llamar a funciones JavaScript
- Podemos asignar nuestros propios eventos a elementos HTML
- Se pueden prevenir eventos
- Y más ...

Cadenas de texto en JavaScript

Las cadenas de texto son usadas para guardar y manipular texto.

Cadenas de texto

Una cadena de texto (o string) simplemente guarda una serie de caracteres. Dentro de una cadena de texto pueden aparecer comillas, siempre que no sean las mismas que lo que rodea a la cadena.

Longitud de una cadena

La longitud de una cadena se puede con la propiedad **length**:

```
var txt = "asdfghjklmn";
var sln = txt.length;
```

Ejemplo:

```
<!DOCTYPE html>
<html>
<body>

<h2>Propiedades de las cadenas de texto en JavaScript</h2>

<p>La propiedad length nos da la longitud de la cadena de texto</p>

<p id="demo"></p>

<script>
var txt = "ABCDEFGHJKLMNOPQRSTUVWXYZ";
document.getElementById("demo").innerHTML = txt.length;
</script>

</body>
</html>
```

Caracteres especiales

Podemos encontrarnos con la necesidad de añadir comillas a nuestro texto, lo que producirá un problema.

```
var y = "Nosotros somos los llamados "Vikingos" del norte."
```

La solución es usar una **caracter de escape** que hace que los caracteres especiales se transformen en caracteres normales. El carácter de escape en JavaScript es la barra invertida (inclinada). \

```
var y = "Nosotros somos los llamados \"Vikingos\" del norte."
```

Ejemplo.

```
<!DOCTYPE html>
<html>
```

```

<body>

<p id="demo"></p>

<script>

var x = "Nosotros somos los llamados \"Vikings\" del norte.";

document.getElementById("demo").innerHTML = x + "<br>";

</script>

</body>
</html>

```

El carácter de escape también se puede usar con otras cosas

| Código | Salida |
|--------|-----------------|
| \' | comilla simple |
| \" | comilla doble |
| \\ | barra invertida |

Existen otros 5 caracteres de escape en JavaScript

| Código | Salida |
|--------|-------------|
| \n | hacia atrás |

saltodecarrosaltodepgina\tabuladorhorizontal\tabuladorvertical

Estos cinco caracteres se desarrollaron originalmente para control de máquinas de escribir, teletipos y máquinas de fax. No tienen ningún sentido en HTML

Quebrando líneas de código largas

Para mejor legibilidad, los programadores suelen no hacer líneas de más de 80 caracteres. Si en JavaScript queremos hacer un salto de línea lo mejor es hacerlo después de un operador:

```

document.getElementById("demo").innerHTML =
"Hola Mundo";

```

También podemos hacer un salto de línea con una barra invertida en una cadena de texto

```

document.getElementById("demo").innerHTML = "Hola \
Mundo";

```


Las cadenas de texto pueden ser objetos

Normalmente las cadenas de texto en JavaScript son valores primitivos. Pero también pueden ser objetos si usamos la palabra clave **new**

```
var x = "Paquito";
var y = new String ("Paquito");

// typeof de x devolverá string
// typeof de y devolverá object
```

Hagamos la prueba

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>

<script>
var x = "Paquito";           // x es una cadena de texto
var y = new String("Paquito"); // y es un objeto

document.getElementById("demo").innerHTML =
typeof x + "<br>" + typeof y;
</script>

</body>
</html>
```

Por lo general esto no se recomienda ya que enlentece la ejecución de la página o script y además puede producir resultados inesperados.

Métodos para cadenas de texto en JavaScript

Los métodos para cadenas de texto nos ayudará a trabajar con ellas

Métodos para cadenas de texto y propiedades

Valores primitivos como "Paquito Chocolatero", no pueden tener ni propiedades ni métodos porque no son objetos. Pero en JavaScript métodos y propiedades están también disponibles para valores, ya que JavaScript trata a los valores primitivos como objetos cuando se ejecutan métodos y propiedades.

Lóngitud de la cadena

Como ya vimos podemos ver la longitud de una cadena con la propiedad **length**

```
var txt = "abcdefg";  
var sln = txt.length;
```

Encontrando una cadena en una cadena

El método **IndexOf()** devuelve el índice de (la posición) de la **primera** ocurrencia de un texto especificado en una cadena.

Ejemplo

```
var str = "Localizamos el siguiente texto!";  
var pos = str.indexOf("siguiente");
```

El método **lastIndexOf()** devuelve el índice de la última ocurrencia de un texto especificado en una cadena.

Ejemplo:

```
var str = "Localizamos el siguiente texto!";  
var pos = str.lastIndexOf("siguiente");
```

Ambos devolverán **-1** si el texto no es encontrado

Además ambos aceptan un segundo parámetro que indica la posición donde se inicia la búsqueda:

```
var str = "Localizamos el siguiente texto!";  
var pos = str.indexOf("siguiente",10);
```

Veamos un ejemplo en una página web:

Buscando una cadena en una cadena

Tenemos un método que nos permite buscar una cadena para un determinado valor y nos devuelve la posición de donde lo ha encontrado:

Ejemplo:

```
var str = "Localizamos el siguiente texto!";  
var pos = str.search("siguiente");
```

La pregunta ahora es evidente: ¿los métodos **indexOf** y **search()** son iguales?

Estos métodos son bastantes parecidos pero tienen algunas diferencias:

- El método **search()** no admite un segundo argumento
- El método **search()** tiene mucho más poder de búsqueda (admite expresiones regulares)

Extrayendo partes de texto

Hay tres métodos para extraer partes de texto de una cadena:

1. slice (comienzo, final)
2. substring (comienzo, final)
3. substr (comienzo, longitud)

- El método slice()

slice() extrae una parte de una cadena de texto y devuelve la parte extraída como una nueva cadena de texto. El método toma de dos argumentos que son la posición inicial y la final.

Ejemplo:

```
var str = "Manzana, Plántano, Kiwi";
var res = str.slice(9,16)
```

Si ponemos parámetros negativos la posición será contada desde el final al principio.

Ejemplo:

```
<!DOCTYPE html>
<html>
<body>

<h2>Métodos para cadenas de texto</h2>

<p id="demo"></p>

<script>
var str = "Manzana, Plántano, Kiwi";
var res = str.slice(9,16);
document.getElementById("demo").innerHTML = res;
</script>

</body>
</html>
```

Si omitimos el segundo parámetro cortaremos desde la posición indicada hasta el final

```
var res = str.slice(9);
```

- El método `substring()`

El método `substring` es igual al método `slice()` sólo que no puede aceptar valores **negativos**.

- El método `substr()`

Es muy parecido al método `slice()` sólo que el segundo parámetro es la longitud de la cadena que queremos obtener.

Reemplazando texto

El método `replace()` reemplaza un texto por otro.

Ejemplo:

```
str = "Por favor visita la web: www.google.es";
var n = str.replace("www.google.es", "www.maxxcan.com");
```

Por defecto, `replace()` solo reemplaza la primera concurrencia que encuentre. Para hacerlo con más elementos tenemos que usar una **expresión regular** con `/g`.

Por defecto, `replace()` es case sensitivo, distingue entre mayúsculas y minúsculas para evitar esto usaremos la expresión regular `/i`

Ejemplos:

```
str = "El mejor sistema operativo es Windows y Windows";
var n = str.replace(/Windows/g, "Linux");

str = "El mejor sistema operativo es Windows";
var n = str.replace(/WINDOWS/i, "Linux");
```

Convirtiendo mayúsculas en minúsculas y viceversa

Esto se hace con los métodos `toUpperCase()` y `toLowerCase()`:

```
var texto1 = "Hola, Mundo!";
var texto2 = texto1.toUpperCase();

var texto1 = "Hola, Mundo!";
var texto2 = texto1.toLowerCase();
```

Concatenar texto

Para ello tenemos el método **concat()**

```
var texto1 = "Hola";  
var texto2 = "Mundo";  
var text3 = text1.concat(" ", texto2);
```

Podemos usarlo en vez del operador **+**.

Todos estos métodos crean una nueva cadena de texto, lo que quiere decir que no altera el original.

Extrayendo caracteres

Para extraer caracteres tenemos dos métodos:

1. **charAt(posición)**
2. **charCodeAt(posición)**

- El método **charAt()**

El método **charAt()** devuelve el carácter de la posición especificada. Recordar que las posiciones empiezan por 0.

Ejemplo

```
var str = "Hola Mundo";  
str.charAt(0); // esto devuelve la H
```

- El método **charCodeAt()**

Este método nos devuelve el unicode del carácter especificado.

Ejemplo

```
var str = "Hola Mundo";  
str.charCodeAt(0); // esto nos devuelve 72
```

Números en JavaScript

En JavaScript sólo hay un tipo de número. Los números pueden ser escritos con o sin decimales.

```
var x = 3.14;  
var y = 3;
```

También podemos usar notación científica

```
var x = 123e5;  
var y = 123e-5;
```

Precisión

Los números enteros tienen una precisión de 15 dígitos

```
var x = 999999999999999; // x será 999999999999999
var y = 999999999999999; // y será 1000000000000000
```

El máximo número de decimales es 17.

———WARNING———

JavaScript usa el operador + para adición y concatenación. Los números son añadidos y las cadenas de texto concatenadas.

```
var x = 10;
var y = 20;
var z = x + y; // z valdrá 30

var x = "10";
var y = "20";
var z = x + y; // z valdrá 1020
```

Cuidado con esto porque además con que uno de las variables sea una cadena de texto lo serán los dos.

```
<!DOCTYPE html>
<html>
<body>

<h2>Números en JavaScript</h2>

<p>Qué esperamos aquí?</p>

<p id="demo"></p>

<script>
var x = 10;
var y = 20;
document.getElementById("demo").innerHTML =
"The result is: " + x + y;
</script>

</body>
</html>
```

y aquí:

```
<!DOCTYPE html>
<html>
<body>

<h2>Números en JavaScript</h2>

<p>Y aquí?</p>

<p id="demo"></p>

<script>
var x = 10;
var y = 20;
var z = "30";
var result = x + y + z;
document.getElementById("demo").innerHTML = result;
</script>

</body>
</html>
```

Números en cadenas de texto

Las cadenas de texto en JavaScript pueden tener caracteres numéricos

```
var x = "2017";
```

JavaScript intentará convertir la cadena a números en operaciones numéricas

```
var x = "100";
var y = "10";
var z = x / y; // z será 10
```

Esto funciona con la división, resta y la multiplicación pero no con la suma.

```
var x = "100";
var y = "10";
var z = x + y; // z no será 110 sino 10010
```

NaN - Not a Number o no es un números

Esto JavaScript se lo reserva para cuando un número no es un número legal o válido.

Ejemplo:

```
var x = 100 / "Manzanas"; // x será un NaN
```

Además tenemos el método **isNaN()** para ver si un número es un NaN o no.

Infinito

Infinito (o -infinito) es el valor que JavaScript devolverá si calculamos un número que sea mayor de lo posible.

Hexadecimal

JavaScript interpreta números como hexadecimales si van precedidos por un Ox

Ejemplo:

```
var x = 0xFF; // x será 255
```

Por defecto, JavaScript muestra los números en base 10, pero podemos pasarlos a hexadecimales (base 16), octales (base 8) y binarios (base 2).

Los números pueden ser objetos

Un número se transforma en objeto si para definirlo usamos la palabra clave **new**

Métodos para números

El método toString()

toString() devuelve un número como una cadena de texto.

```
var x = 123;  
x.toString();  
(123).toString();  
(100 + 23).toString();
```


El método toExponential()

toExponential() devuelve una cadena con un número redondeado usando notación científica.

Un parámetro define el número de caracteres detrás del punto decimal

```
var x = 9.656;
x.toExponential(2); // devuelve 9.66e+0
x.toExponential(4); // devuelve 9.6560e+0
x.toExponential(6); // devuelve 9.656000e+0
```

El parámetro es opcional y si no se especifica no se hará el redondeo.

El método toFixed()

toFixed() devuelve una cadena con el número escrito con un número de decimales especificado.

```
var x = 9.656;
x.toFixed(0); // devuelve 10
x.toFixed(2); // devuelve 9.66
x.toFixed(4); // devuelve 9.6560
x.toFixed(6); // devuelve 9.656000
```

El método toPrecision()

toPrecision() devuelve una cadena con un número escrito con una determinada longitud.

```
var x = 9.656;
x.toPrecision(); // devuelve 9.656
x.toPrecision(2); // devuelve 9.7
x.toPrecision(4); // devuelve 9.656
```

El método valueOf()

valueOf() devuelve un número como un número

```
var x = 123;
x.valueOf(); // devuelve 123 de la variable x
```

En javascript, un número puede ser un valor primitivo o un objeto. El método valueOf() es usado internamente para convertir Números objeto a valores primitivos.

Convirtiendo variables a números

Hay 3 métodos en JavaScript que pueden ser usados para convertir variables a números:

1. El método `number()`
 2. El método `parseInt()`
 3. El método `parseFloat()`
- El método `number()`
 - El método `parseInt()`
 - El método `parseFloat()`

Matemáticas en JavaScript

En JavaScript tenemos una serie de objetos que nos permiten hacer operaciones matemáticas sobre los números.

Math.round()

Para redondear números

```
Math.round(4.7) // devuelve 5  
Math.round(4.4) // devuelve 4
```

Math.pow()

Para hacer potencias de x a la potencia de y . `Math.pow(x, y)`

```
Math.pow(8, 2) // devuelve 64
```

Math.sqrt()

Para hacer raíces

```
Math.sqrt(64) // devuelve 8
```

Math.abs()

Devuelve el valor absoluto

```
Math.abs(-38) // devuelve 38
```

Math.ceil

Devuelve el redondeo hacia arriba

```
Math.ceil(4.4); // devuelve 5
```

Math.floor()

Devuelve el redondeo hacia abajo

Math.sin() y Math.cos()

Devuelve el seno y coseno de un número

Math.random()

Devuelve un número al azar

Constantes

En JavaScript tenemos varias constantes matemáticas accesibles

Ejemplo

```
Math.E()  
Math.PI()  
Math.LN2()
```

Fechas en JavaScript

Vamos a ver algo muy importante que es el cómo JavaScript trabaja con las fechas

Formatos

Las fechas pueden ser escritas como cadenas de texto:

Fri Jul 14 2017 12:01:28 GMT+0200 (CEST)

O como un número:

1500026488420

Mostrando fechas

La forma más sencilla de mostrar una fecha es con el método `Date()`.

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = Date();
</script>
</body>
</html>
```

Creando un objeto fecha

Los objeto fechas nos permite trabajar con las fechas

Las fechas están compuestas de años, meses, días, horas, minutos, segundos y milisegundos.

Los objetos fecha se crean con el constructor **new Date()** y hay 4 formas de hacerlo.

```
new Date()
new Date(milliseconds)
new Date(dateString)
new Date(year, month, day, hours, minutes, seconds, milliseconds)
```

Usando `new Date()`, creamos un nuevo objeto con la **fecha actual**

```
<script>
var d = new Date();
document.getElementById("demo").innerHTML = d;
</script>
```

Si usamos `new Date(date string)`, creamos un objeto con una fecha específica

```
<script>
var d = new Date("October 13, 2014 11:13:00");
document.getElementById("demo").innerHTML = d;
</script>
```

También podemos usar otras formas de entrada como números.

Métodos para fechas

Una vez creado el objeto tenemos métodos que te permiten operar con él.

Mostrando fechas

Cuando mostramos un objeto fecha en HTML, este es automáticamente convertido a cadena de texto, con el método **toString**.

Así esto:

```
<p id="demo"></p>

<script>
d = new Date();
document.getElementById("demo").innerHTML = d;
</script>
```

Es lo mismo que esto:

```
<p id="demo"></p>

<script>
d = new Date();
document.getElementById("demo").innerHTML = d.toString();
</script>
```

También tenemos el método **toUTCString()** que convierte la fecha a UTC.

```
<script>
var d = new Date();
document.getElementById("demo").innerHTML = d.toUTCString();
</script>
```

Además tenemos el método **toDateString()** que convierte la fecha a un formato más legible.

```
var d = new Date();
document.getElementById("demo").innerHTML = d.toDateString();
```

Probémoslo:

Formato en las fechas en JavaScript

Hay 4 tipos de formatos de entrada para las fechas en JavaScript

| Tipo | Ejemplo |
|------------|---|
| ISO Date | "2017-06-17" (Es el estándar internacional) |
| Short Date | "17/06/2017" |
| Long Date | "Jul 17 2017" o "17 Jul 2017" |
| Full Date | "Monday July 17 2017" |

El formato ISO sigue el estándar de forma estricta. Los otros dependen más del navegador.

Métodos en Fechas en JavaScript

Los métodos en fechas nos permiten obtener y establecer valores para las fechas (años, meses, días, horas, minutos, segundos y milisegundos)

Métodos Get

Los métodos **Get** nos permiten obtener datos de la fecha. Aquí están los más comunes:

| Método | Descripción |
|-------------------|---|
| getDate() | Toma el día como un número |
| getDay() | Toma el día de la semana como un número |
| getFullYear() | Toma los 4 dígitos del año |
| getHours() | Toma la hora |
| getMilliseconds() | toma los milisegundos |
| getMinutes() | Toma los minutos |
| getMonth() | Toma los meses |
| getSeconds() | Toma los segundos |
| getTime() | Toma el tiempo (milisegundos desde el 1 de Enero de 1970) |

Ejemplo:

```
var d = new Date();  
document.getElementById("demo").innerHTML = d.getTime();
```

Métodos Set

Son métodos para establecer una parte de la fecha.

Ejemplo:

```
var d = new Date();  
d.setFullYear(2020, 0, 14);  
document.getElementById("demo").innerHTML = d;
```

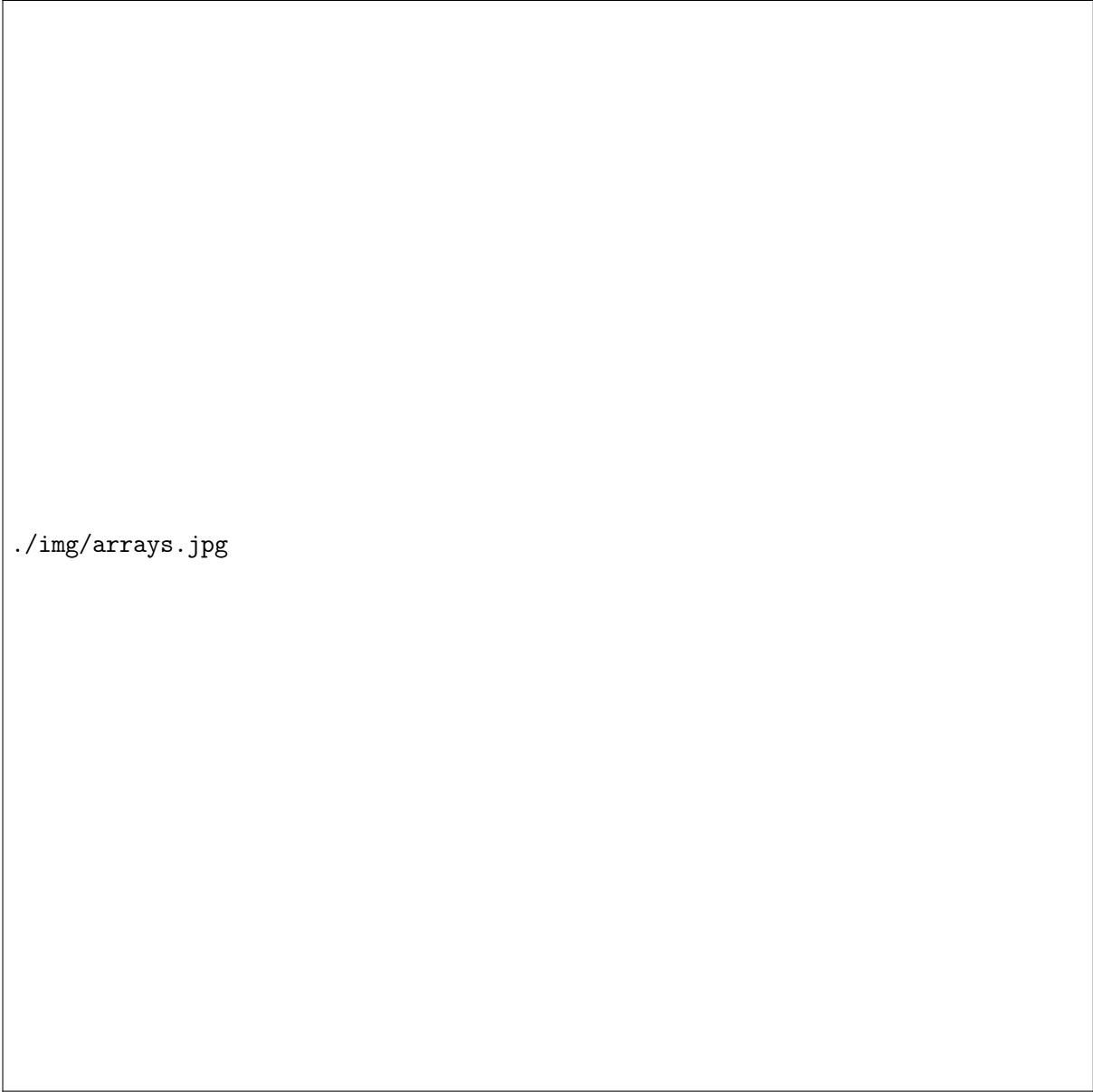
Comparando fechas

Las fechas pueden ser comparadas fácilmente. Veamos el siguiente ejemplo:

```
var hoy, algunDia, texto;
hoy = new Date();
algunDia = new Date();
algunDia.setFullYear(2017, 09, 01);

if (algunDia > hoy) {
    texto = "Hoy es antes del apocalipsis final";
} else {
    texto = "Hoy es después del fallido apocalipsis final, más suerte la próxima vez";
}
```

Arrays en JavaScript



./img/arrays.jpg

Un array es usado para guardar múltiples valores en una simple variable.

```
var coches = ["Volvo", "BMW", "Seat"];
```


Qué es un array

Un array es una variable especial la cual puede contener más de un valor a la vez. Si tienes una lista de artículos, podríamos guardarlos en variables.

```
var coche1 = "Volvo";  
var coche2 = "BMW";  
var coche3 = "Seat";
```

Como vemos esto es un poco complicado. La solución son los Arrays.

Creando arrays

La forma fácil es:

```
var nombre_array = [artículo1, artículo2, ...];
```

Un array podemos declararlo en una sola línea o en varias.

```
var coches = ["Volvo", "BMW", "Seat"];
```

ó

```
var coches = [  
    "Volvo";  
    "BMW";  
    "Seat";  
]
```

Usando la palabra clave new

También podemos usar la palabra clave new para crear nuevos arrays

```
var coches = new Array("Volvo", "BMW", "Seat");
```

Esta forma tiene el mismo resultado que el anterior.

Accediendo a un elemento del array

Cada elemento del array tiene un índice por así decirlo. El índice empieza por 0 así que para acceder a un elemento del array hay que poner la variable y el índice entre corchetes.

```
var coches = ["Volvo", "BMW", "Seat"];  
document.getElementById("demo").innerHTML = coches[0];
```

Los arrays son objetos

Los arrays son un tipo especial de objeto. Si utilizamos el operador **typeof** con un array nos devolverá "object".

Los elementos de un array pueden ser objetos

En JavaScript podemos tener objetos en un array, pero también funciones e incluso arrays en un array.

Propiedades en los arrays

La fuerza real de un array en JavaScript son las propiedades y métodos que éstos tienen.

| Propiedades | Descripción |
|-------------|--|
| constructor | Devuelve una función que crea un prototipo de objeto Array |
| length | Devuelve el número de elementos que hay en un array |
| prototype | Permite añadir propiedades y métodos a un objeto Array |

Añadiendo elementos al arrays

Hay dos formas de añadir elementos a un array

- Método push
- Método lenght

Ejemplos:

```
var coches = ["Volvo", "BMW", "Seat"];
coches.push("Mercedes");
```

```
var coches = ["Volvo", "BMW", "Seat"];
coches[coches.length] = "Mercedes";
```

Arrays asociativos

En muchos lenguajes se puede hacer que los arrays en vez de estar asociados a números se asocien a nombres. Estos se llaman arrays asociativos o hashes. En JavaScript esto no es posible

Diferencia entre un array y un objeto

- Los arrays usan índices numerados
- Los objetos usan índices nominales
- Cuando usar uno u otro

Pues como en JavaScript no hay arrays asociativos tendremos que usar objetos cuando queramos algo como eso.

Métodos en los arrays de JavaScript

- Convertir arrays a cadenas de texto

Para eso tenemos el método **toString()**

Ejemplo:

```
var frutas = ["Plátanos", "Naranja", "Manzana"];
document.getElementById("demo").innerHTML = frutas.toString();
```

Resultado:

Esto realmente no hace falta porque JavaScript lo hace automáticamente.

- Otros métodos

Otros métodos muy interesantes son:

| Método | Descripción |
|---------|--|
| pop() | Elimina el último elemento del array y lo devuelve |
| push() | Añade un nuevo elemento al array |
| shift() | Elimina el primer elemento del array y lo devuelve |
| delete | Elimina un elemento |
| ... | |

- Ordenando arrays

Para ordenar arrays tenemos el método **sort()**

Ejemplo:

```
var frutas = ["Plátanos", "Naranja", "Manzana"];
document.getElementById("demo").innerHTML = frutas.sort();
```

También podemos hacer el ordenamiento inverso con **reverse()**

```
var frutas = ["Plátanos", "Naranja", "Manzana"];
document.getElementById("demo").innerHTML = frutas.reverse();
```

Booleanos

Un booleano es algo así como un sí o un no, un verdadero o falso.

Podemos usar la función `Boolean()` para obtener booleanos `ç`

```
Boolean(10 > 9)
```

o más fácil

```
(10 > 9)
```

```
10 > 9
```

En un ejemplo:

```
<!DOCTYPE html>
<html>
<body>

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>
function myFunction() {
    document.getElementById("demo").innerHTML = 10 > 9;
}
</script>

</body>
</html>
```

Comparación y operadores lógicos en JavaScript

La comparación y los operadores lógicos son usados para los test de verdad o falso.

Operadores de Comparación

Los operadores de comparación son usados dictados lógicos para determinar la igual o diferencia entre variables o valores.

Presuponiendo que `x = 5`

| Operador | Descripción | Comparando | Devolución |
|----------|--------------------------|------------|------------|
| == | igual a | x == 8 | false |
| = | igual valor y tipo | x = 5 | true |
| != | no igual | x != 8 | true |
| !== | no igual de valor y tipo | x !== 5 | false |
| > | mayor que | x > 8 | false |
| < | menor que | x < 8 | true |
| >= | mayor o igual que | x >= 8 | false |
| <= | menor o igual que | x <= 8 | true |

- Cómo puede ser usados

Pues normalmente se usan en declaraciones de condición que compara valores y depende del resultado toma una acción u otra.

Ejemplo:

```
if (edad < 18) texto = "Demasiado joven";
```

Operadores lógicos

Los operadores lógicos son usados para determinar la lógica entre variables o valores.

Dado x = 6 e y = 3, veamos la siguiente tabla:

| Operador | descripción | Ejemplo |
|----------|-------------|-------------------------|
| && | y | (x <10 && y >1) is true |
| | | |

Operador condicionales

JavaScript también contiene un operador condicional que asigna un valor a una variable basado en alguna condición.

Sintaxis

```
nombreVariable = (condicion) ? valor1:valor2
```

Ejemplo:

```
<!DOCTYPE html>
<html>
<body>
```

```
<p>Introduce tu edad pulsa el botón:</p>
```

```

<input id="age" value="18" />

<button onclick="myFunction()">Púlsame</button>

<p id="demo"></p>

<script>
function myFunction() {
    var edad, votable;
    edad = document.getElementById("age").value;
    votable = (edad < 18) ? "Demasiado Joven":"Edad suficiente";
    document.getElementById("demo").innerHTML = votable + " para votar.";
}
</script>

</body>
</html>

```

Condicionales en JavaScript

Las declaraciones condicionales son para realizar diferentes acciones según diferentes condiciones.

Condicionales

En JavaScript existen los siguientes condicionales:

- **If** para especificar que un bloque se ejecutará si la condición es verdadera
- **Else** para especificar que un bloque se ejecutará si la condición es falsa
- **Else if** para especificar una nueva condición si la primera condición es falsa
- **Switch** para especificar muchos bloques de código alternativos para ser ejecutados

La declaración if

El uso de if es para especificar que un bloque de JavaScript puede ser ejecutado si una condición es verdadera.

```

if (condición) {
    bloque de código a ser ejecutado si la condición es verdadera
}

```

Ejemplo:

```
<!DOCTYPE html>
<html>
<body>

<p>Muestra "Buenos días!" si la hora es menos de las 12:00:</p>

<p id="demo">Buenas tardes</p>

<script>
if (new Date().getHours() < 12) {
    document.getElementById("demo").innerHTML = "Buenos días!";
}
</script>

</body>
</html>
```

La declaración else

Con **else** especificamos otro bloque de código que se ejecutará si la condición es falsa

```
if (condición){
    bloque de código que se ejecutará si la condición es verdadera
} else {
    bloque de código que se ejecutará si la condición es falsa
}
```

Ejemplo:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
</head>
<body>

<p>Pulsa el botón para que muestre el mensaje correspondiente:</p>

<button onclick="myFunction()">Púlsame</button>
```

```

<p id="demo"></p>

<script>
function myFunction() {
    var hora = new Date().getHours();
    var saludo;
    if (hora < 13) {
        saludo = "Buenos días";
    } else {
        saludo = "Buenas tardes";
    }
    document.getElementById("demo").innerHTML = saludo;
}
</script>

</body>
</html>

```

La declaración else if

Con else if especificamos una nueva condición si la primera condición es falsa.

```

if (condición 1){
    bloque de código que será ejecutado si la condición es cierta
} else if (condición 2){
    bloque de código que será ejecutado si la condición 1 es falsa y la condición 2 es cierta
} else {
    bloque de código que será ejecutado si la condición 1 y 2 son falsas
}

```

Ejemplo

```

<!DOCTYPE html>
<html>
<body>

<p>Pulsa el botón para que te salude</p>

<button onclick="myFunction()">Púlsame</button>

```



```

<p id="demo"></p>

<script>
function myFunction() {
    var greeting;
    var time = new Date().getHours();
    if (time < 10) {
        greeting = "Buenos días";
    } else if (time < 12) {
        greeting = "Buenas tardes";
    } else {
        greeting = "Buenas noches";
    }
    document.getElementById("demo").innerHTML = greeting;
}
</script>

</body>
</html>

```

La declaración Switch

Es para seleccionar un bloque de código de entre varios para ser ejecutado.

```

switch (expresión) {
    caso n:
        bloque de código
        break;
    caso n:
        bloque de código
        break;
    por defecto:
        bloque de código
}

```

Cómo trabaja esto:

1. La expresión es evaluada una vez
2. El valor de la expresión es comparado con los valores en cada caso
3. Si hay coincidencia, el bloque asociado será ejecutado

Ejemplo:

Vamos a ver en qué día de la semana estamos:

```
<!DOCTYPE html>
<html>
<body>

<p>Pulsa el botón para saber el día de la semana</p>

<button onclick="myFunction()">Púlsame</button>

<p id="demo"></p>
<script>
var day;
switch (new Date().getDay()) {
    case 0:
        day = "Domingo";
        break;
    case 1:
        day = "Lunes";
        break;
    case 2:
        day = "Martes";
        break;
    case 3:
        day = "Miércoles";
        break;
    case 4:
        day = "Jueves";
        break;
    case 5:
        day = "Viernes";
        break;
    case 6:
        day = "Sábado";
}
document.getElementById("demo").innerHTML = "Hoy es " + day;
</script>

</body>
```

</html>

- La palabra clave break

Cuando JavaScript se encuentra con una palabra clave *break, sale del bloque switch

- La palabra clave default

Especifica qué código correr si no se encuentra ningún caso:

```
switch (new Date().getDay()){
    case 6:
        day = "Sabado";
        break;
    default:
        texto = "Mirando para otro día";
}
```

El caso default no tiene por qué ir el último.

- Bloque común a varios bloques

Si varios bloques tienen el mismo código se pueden unir:

```
switch (new Date().getDay()){
    case 1:
    case 2:
    case 3:
    case 4:
    case 5:
        texto = "Estamos en la semana";
        break;
    case 6:
    case 0:
        texto = "Por fin es fin de semana";
}
```

Ejercicio. Quiero que a determinadas marcas de coches me diga la nacionalidad de este:

```
<!DOCTYPE html>
<html>
<body>
```

```

<input id="myInput" type="text" value="BMW">
<button onclick="checkCar()">Comprueba el coche</button>
<p id="demo"></p>

<script>
function checkCar() {
    var text;
    var favCar = document.getElementById("myInput").value;

    switch(favCar) {
        case "BMW":
            text = "Coche Aleman";
            break;
        case "Ford":
            text = "Coche americano";
            break;
        case "Peugeot":
            text = "coche francés";
            break;
        default:
            text = "nacionalidad desconocida";
    }
    document.getElementById("demo").innerHTML = text;
}
</script>

</body>

```

Bucles for

Un bucle un un bloque de código que se repite un número de veces.

Con los bucles nosotros podemos ejecutar un código una y otra vez pero cada vez con un valor diferente

Por ejemplo:

En vez de escribir esto:

```

text += cars[0] + "<br>";
text += cars[1] + "<br>";
text += cars[2] + "<br>";
text += cars[3] + "<br>";
text += cars[4] + "<br>";

```

```
text += cars[5] + "<br>";
```

Podemos escribir esto:

```
for (i = 0; i < cars.length; i++) {  
    text += cars[i] + "<br>";  
}
```

Ejercicio.

Vamos a hacer un listado de coches: BMW, Volvo, Ford, Audi, Mercedes y Seat

```
<!DOCTYPE html>  
<html>  
<body>  
  
<h2>Bucles en JavaScript</h2>  
  
<p id="demo"></p>  
  
<script>  
var cars = ["BMW", "Volvo", "Ford", "Audi", "Mercedes", "Seat"];  
var text = "";  
var i;  
for (i = 0; i < cars.length; i++) {  
    text += cars[i] + "<br>";  
}  
document.getElementById("demo").innerHTML = text;  
</script>  
  
</body>  
</html>
```

Diferentes tipos de bucles

JavaScript soporta diferentes tipos de bucles:

- for. El bucle ejecuta un bloque de código un número de veces
- for/in. El bucle ejecuta las propiedades de un objeto
- while. El bucle ejecuta el bloque de código mientras una condición sea cierta
- do/while. igual que la anterior

Bucle For

El bucle for es la herramienta que solemos usar para crear un bucle.

La sintaxis es

```
for (declaración 1; declaración 2; declaración 3) {  
  bloque de código que se ejecuta  
}
```

- La declaración 1 se ejecuta antes de que el bucle comience.
- La declaración 2 define la condición para ejecutar el bucle
- La declaración 3 una vez que el bucle ha finalizado

Ejemplo:

```
for (i = 0; i < 5; i++){  
  texto += "El número es " + i + "<br>";  
}
```

Ejemplo html

```
<!DOCTYPE html>  
<html>  
<body>  
  
<h2>Bucles en JavaScript</h2>  
  
<p id="demo"></p>  
  
<script>  
var text = "";  
var i;  
for (i = 0; i < 5; i++) {  
  text += "El número es " + i + "<br>";  
}  
document.getElementById("demo").innerHTML = text;  
</script>  
  
</body>  
</html>
```

- Declaración 1.

Normalmente usamos la declaración 1 para inicializar la variable usada en el bucle. Si no fuera necesario no hay problema para JavaScript ya que es opcional.

Además podemos iniciar muchos valores en la declaración 1 separados por comas.

Ejemplo:

```
<!DOCTYPE html>
<html>
<body>

<p id="demo"></p>

<script>
var coches = ["BMW", "Volvo", "Saab", "Ford"];
var i, len, texto;
for (i = 0, len = cars.length, texto = ""; i < len; i++) {
    texto += coches[i] + "<br>";
}

document.getElementById("demo").innerHTML = texto;
</script>

</body>
</html>
```

Podemos omitir la declaración 1.

Ejemplo:

```
<!DOCTYPE html>
<html>
<body>

<p id="demo"></p>

<script>
var coches = ["BMW", "Volvo", "Saab", "Ford"];
var i = 2;
var len = cars.length;
var texto = "";
```

```

for (; i < len; i++) {
    texto += coches[i] + "<br>";
}

document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>

```

- Declaración 2

La declaración 2 es la condición para que se ejecute el bucle y este se ejecutará mientras la condición sea verdadera. Si no hay condición, la declaración 2 podemos eliminarla ya que es opcional, pero si la eliminamos tendremos que añadir un **break** dentro del bucle o este se repetirá infinitamente.

- Declaración 3

Usamos la declaración 3 para incrementar el valor de la variable inicial. Podemos añadir un incremento positivo o negativo.

También es opcional (pero yo no recomiendo quitarlo)

Bucle For/Indefinido

El bucle lo hace pero a través de las propiedades de un objeto:

```

<!DOCTYPE html>
<html>
<body>

<h2>Bucles en JavaScript</h2>

<p id="demo"></p>

<script>
var txt = "";
var person = {fname:"Paquito", lname:"Chocolatero", age:100};
var x;
for (x in person) {
    txt += person[x] + " ";
}

```



```

}
document.getElementById("demo").innerHTML = txt;
</script>

</body>
</html>

```

Bucle While

Los bucles while ejecutan un código mientras una condición sea verdadera
 Sintaxis:

```

while (condición){
  bloque de código a ejecutar
}

```

Ejemplo:

```

<!DOCTYPE html>
<html>
<body>

<h2>Bucle While</h2>

<p id="demo"></p>

<script>
var texto = "";
var i = 0;
while (i < 10) {
  texto += "<br>El número es " + i;
  i++;
}
document.getElementById("demo").innerHTML = texto;
</script>

</body>
</html>

```

Si olvidamos poner el incremento el bucle nunca acabará y colgará tu navegador

Bucle Do/While

Es una variante del while y lo que hace es ejecutar una vez el código y si la condición es verdadera lo volverá a hacer mientras la condición siga siendolo.

Sintaxis:

```
do {  
    código a ejecutar  
}  
while (condición);
```

Ejemplo:

```
<!DOCTYPE html>  
<html>  
<body>  
  
<h2>do ... while</h2>  
  
<p id="demo"></p>  
  
<script>  
var texto = ""  
var i = 0;  
  
do {  
    texto += "<br>El número es " + i;  
    i++;  
}  
while (i < 10);  
  
document.getElementById("demo").innerHTML = text;  
</script>  
  
</body>  
</html>
```

Comparando For y While

Vamos a ver dos bucles:

```

<!DOCTYPE html>
<html>
<body>

<p id="demo"></p>

<script>
var coches = ["BMW", "Volvo", "Saab", "Ford"];
var i = 0;
var texto = "";
for (;coches[i];) {
    texto += coches[i] + "<br>";
    i++;
}
document.getElementById("demo").innerHTML = text;
</script>

</body>

```

y el segundo:

```

<!DOCTYPE html>
<html>
<body>

<p id="demo"></p>

<script>
var coches = ["BMW", "Volvo", "Saab", "Ford"];
var i = 0;
var texto = "";
while (coches[i]) {
    texto += coches[i] + "<br>";
    i++;
}
document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>

```

Ejercicios

Haced una lista de números que vaya de 0 a 10. Con for, con while y con do/while

Break y Continue

- Break

La declaración Break sirve para salir de un bucle, ya lo vimos anteriormente para salir de un switch(). Break rompe el bucle y continúa ejecutando el código que hay después del bucle (si es que lo hay)

Ejemplo:

```
<!DOCTYPE html>
<html>
<body>

<p>Un bucle con un break</p>

<p id="demo"></p>

<script>
var text = "";
var i;
for (i = 0; i < 10; i++) {
    if (i === 3) { break; }
    text += "El número es " + i + "<br>";
}
document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

- Continue

La declaración continue rompe una iteración en el bloque si una condición especificada ocurre y continua con la próxima iteración del bucle.

Ejemplo:

```
<!DOCTYPE html>
<html>
<body>
```

```

<p>El bucle se saltará un paso cuando i = 3.</p>

<p id="demo"></p>

<script>
var text = "";
var i;
for (i = 0; i < 10; i++) {
    if (i === 3) { continue; }
    text += "El número es " + i + "<br>";
}
document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>

```

Formularios en JavaScript

Validación de formularios en JavaScript

Un formulario HTML puede ser validado por JavaScript

Ejemplo

```

function validateForm() {
    var x = document.forms["myForm"]["fnombre"].value;
    if (x == ""){
        alert("El campo nombre debe ser rellenado");
        return false;
    }
}

```

Esta función puede ser llamada cuando el formulario sea enviado:

```

<form name="myForm" action="/action_page_post.php" onsubmit="return validateForm()" method="post">
Name: <input type="text" name="fnombre">
<input type="submit" value="Submit">
</form>

```

Ejercicio: Validar que algunos campos de un formulario son rellenados.

```

<!DOCTYPE html>
<html>
<head>
<script>
function validateForm() {
    var x = document.forms["myForm"]["fnombre"].value;
    if (x == "") {
        alert("El campo nombre debe ser rellenado");
        return false;
    }
}
</script>
</head>
<body>

<form name="myForm" action="/action_page_post.php"
onsubmit="return validateForm()" method="post">
Nombre: <input type="text" name="fnombre">
<input type="submit" value="Enviar">
</form>

</body>
</html>

```

Un script más complejo nos valdría para ver si un campo rellenado es un número.

```

<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
</head>
<body>

<h2>Validación en HTML con JavaScript</h2>

<p>Introduce un número entre 1 y 10:</p>

<input id="numb">

<button type="button" onclick="myFunction()">Enviar</button>

```

```

<p id="demo"></p>

<script>
function myFunction() {
    var x, text;

    x = document.getElementById("numb").value;

    if (isNaN(x) || x < 1 || x > 10) {
        text = "No es una entrada válida";
    } else {
        text = "Te doy el apruvement";
    }
    document.getElementById("demo").innerHTML = text;
}
</script>

</body>
</html>

```

Validación automática de formularios

Los formularios pueden ser automáticamente por el navegador. Para ello ponemos en el campo que no queremos que quede vacío el atributo de **required**

Ejemplo:

```

<form action="/action_page.php" method="post">
<input type="text" name="fnombre" required>
<input type="submit" value="Enviar">
</form>

```

Ejemplo:

```

<!DOCTYPE html>
<html>
<body>

<form action="/action_page_post.php" method="post">

```

```
<input type="text" name="fname" required>
<input type="submit" value="Enviar">
</form>
```

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

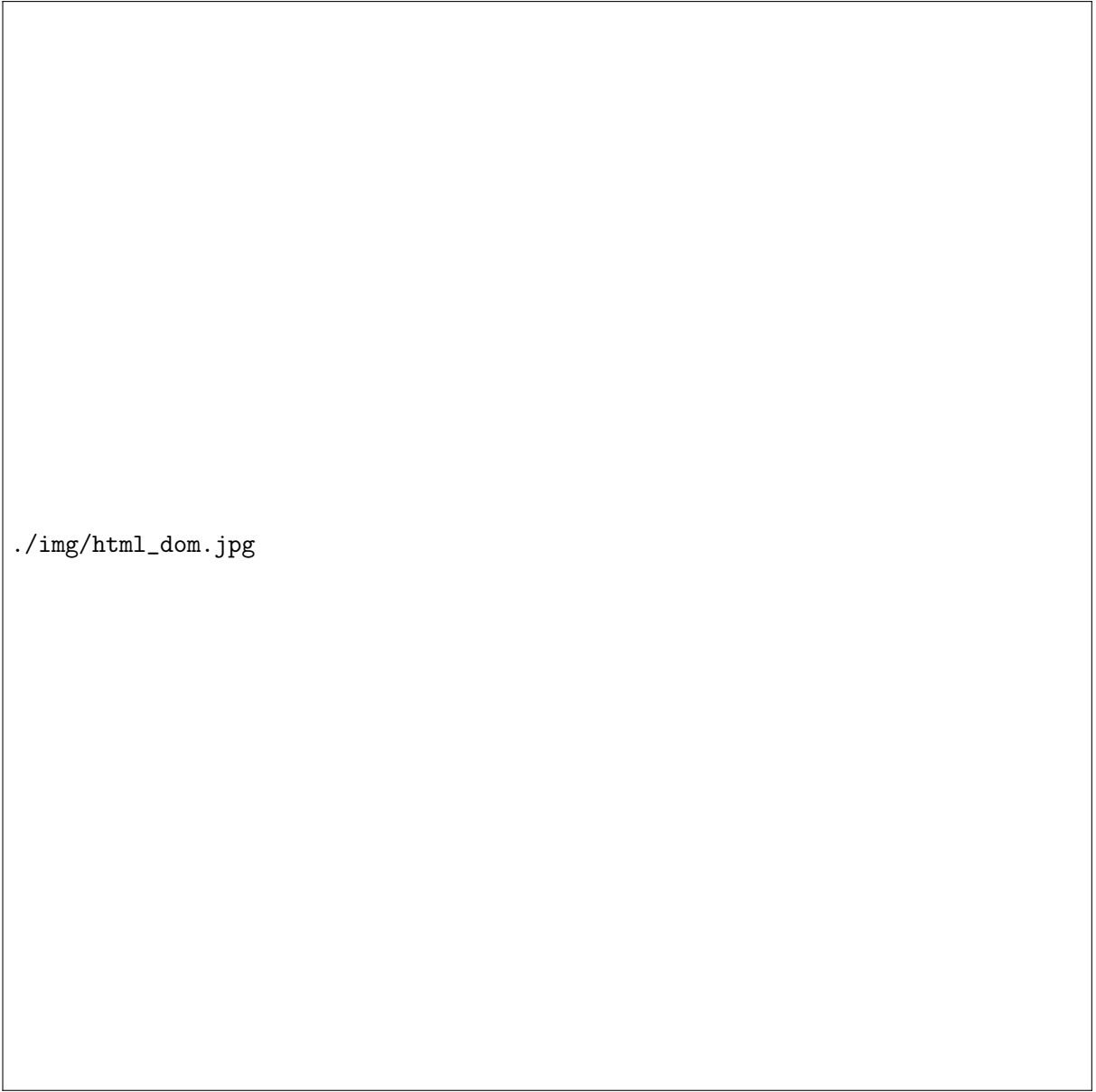
El **problema** es que aún **no funciona** en todos los navegadores

El HTML DOM y JavaScript

El DOM JavaScript puede acceder y cambiar todos los elementos de un documento HTML

Qué es el HTML DOM (Document Object Model)

Cuando una página es cargada, el navegador crea un Documento Modelo de Objetos de la página. El modelo HTML DOM es construido como un árbol de objetos.



`./img/html_dom.jpg`

Con este modelo de objetos, JavaScript obtiene el poder que necesita para crear páginas dinámicas en HTML. Cosas que puede hacer:

- Cambiar los elementos HTML
- Cambiar todos los atributos HTML

- Cambiar el estilo CSS de la página
- Eliminar elementos y atributos existentes
- Añadir nuevos elementos y atributos
- Reaccionar a todos los eventos que ocurran en la página
- Crear nuevos eventos en la páginas

Qué es entonces el DOM

El DOM es un estándar de la W3C (World Wide Web Consortium)

El DOM define un estándar para acceder a los documentos. El W3C DOM está separado en tres partes:

1. Core DOM - modelo estándar para todos los tipos de documentos
2. XML DOM - modelo estándar para todos los documentos XML
3. HTML DOM - modelo estándar para los documentos HTML

Qué es el HTML DOM

Es el modelo estándar de objetos y la interfaz de programación para HTML. El DOM define:

- Los elementos HTML como **objetos**
- Las propiedades de todos los elementos HTML
- Los métodos para acceder a todos los elementos HTML
- Los eventos para todos los elementos HTML

En otras palabras:

El HTML DOM es un estándar de cómo obtener, cambiar, añadir o borrar elementos HTML

Métodos en el HTML DOM

Los métodos en el HTML DOM son **acciones** que pueden desarrollarse sobre elementos HTML. Las propiedades en el HTML DOM son los **valores** (de elementos HTML) que pueden ser establecidos o cambiados.

La interfaz de programación del DOM

El HTML DOM puede ser accesible con JavaScript (y también con otros lenguajes). En el DOM, todos los elementos HTML son definidos como **objetos**.

La **interfaz de programación** son las propiedades y los métodos de cada objeto.

Una **propiedad** es un valor que puede obtenerse o establecerse (como cambiar el contenido de un elemento HTML). Un **método** es una acción que puede hacer (como añadir o borrar un elemento HTML)

```
<html>
<body>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = "Hola Mundo";
</script>

</body>
</html>
```

En este ejemplo:

- getElementById es un **método**
- innerHTML es una **propiedad**
- El método getElementById

Es la forma más común de acceder a un elemento HTML y es usado con el id del elemento. En el ejemplo anterior usamos el id="demo" para encontrar el elemento.

- La propiedad innerHTML

La forma más fácil de obtener el contenido de un elemento. Se utiliza para obtener o reemplazar contenido de elementos HTML.

El Documento HTML DOM

El objeto documento HTML DOM es propio de todos los otros objetos en la página web

Objeto Documento HTML DOM

El objeto documento representa la página web. Si se quiere acceder a cualquier elemento en una página HTML, siempre hay que empezar accediendo al objeto documento. A continuación unos ejemplos de como usar el objeto documento para acceder y manipular HTML

Buscando elementos HTML

| Método | Descripción |
|--|---|
| <code>document.getElementById(id)</code> | Encuentra un elemento por su id |
| <code>document.getElementsByTagName(name)</code> | Encuentra un elemento por su nombre de etiqueta |
| <code>document.getElementsByClassName(name)</code> | Encuentra un elemento por el nombre de su clase |

Cambiando elementos HTML

| Método | Descripción |
|---|--|
| <code>element.innerHTML = nuevo contenido html</code> | Cambia el contenido interno de un elemento |
| <code>element.attribute = nuevo valor</code> | Cambia el valor de un atributo de un elemento html |
| <code>element.setAttribute(atributo, valor)</code> | Cambia el valor de un atributo de un elemento html |
| <code>element.style.property = nuevo estilo</code> | Cambia el estilo de un elemento html |

Añadiendo y eliminando elementos

| Método | Descripción |
|---|------------------------------|
| <code>document.createElement(elemento)</code> | Crea un elemento html |
| <code>document.removeChild(elemento)</code> | Elimina un elemento html |
| <code>document.appendChild(elemento)</code> | Añade un elemento html |
| <code>document.replaceChild(elemento)</code> | Reemplaza un elemento html |
| <code>document.write(texto)</code> | Escribe en la salida de HTML |

Añadiendo eventos manejadores (events handlers)

| Método | Descripción |
|---|--|
| <code>document.getElementById(id).onclick = function(){código}</code> | Añade un evento manejador en un evento onclick |

Encontrando objetos HTML

El primer HTML DOM Nivel 1 (1998), define 11 objetos HTML, colecciones de objetos y propiedades. Todos ellos son válidos en el HTML5. Más tarde, en el HTML DOM Nivel 3, más objetos, colecciones y propiedades fueron añadidas.

| Propiedad | Descripción | DOM |
|------------------------------|--|-----|
| document.anchors | Returns all <a> elements that have a name attribute | 1 |
| document.applets | Returns all <applet> elements (Deprecated in HTML5) | 1 |
| document.baseURI | Returns the absolute base URI of the document | 3 |
| document.body | Returns the <body> element | 1 |
| document.cookie | Returns the document's cookie | 1 |
| document.doctype | Returns the document's doctype | 3 |
| document.documentElement | Returns the <html> element | 3 |
| document.documentMode | Returns the mode used by the browser | 3 |
| document.documentURI | Returns the URI of the document | 3 |
| document.domain | Returns the domain name of the document server | 1 |
| document.domConfig | Obsolete. Returns the DOM configuration | 3 |
| document.embeds | Returns all <embed> elements | 3 |
| document.forms | Returns all <form> elements | 1 |
| document.head | Returns the <head> element | 3 |
| document.images | Returns all elements | 1 |
| document.implementation | Returns the DOM implementation | 3 |
| document.inputEncoding | Returns the document's encoding (character set) | 3 |
| document.lastModified | Returns the date and time the document was updated | 3 |
| document.links | Returns all <area> and <a> elements that have a href attribute | 1 |
| document.readyState | Returns the (loading) status of the document | 3 |
| document.referrer | Returns the URI of the referrer (the linking document) | 1 |
| document.scripts | Returns all <script> elements | 3 |
| document.strictErrorChecking | Returns if error checking is enforced | 3 |
| document.title | Returns the <title> element | 1 |
| document.URL | Returns the complete URL of the document | 1 |

Elementos del HTML DOM

Ahora veremos como encontrar y acceder a elementos HTML en una página HTML.

Buscando elementos HTML

Por supuesto que la gracia de JavaScript es manipular elementos HTML. Pero claro, primero hay que encontrarlos. Hay unas cuantas formas de hacer esto:

- Buscando por su id
- Buscando por el nombre de su etiqueta
- Por su nombre de clase

- Por su selector CSS
- Por su colección de objetos HTML

Buscando elementos HTML por su id

Es la forma más sencilla de encontrar un elemento en el DOM.

Ejemplo:

```
var miElemento = document.getElementById("demo");
```

Ejemplo más complejo:

```
<!DOCTYPE html>
<html>
<body>

<p id="intro">Hola Mundo!</p>

<p>Ejemplo de cómo funciona <b>getElementById</b></p>

<p id="demo"></p>

<script>
var miElemento = document.getElementById("intro");
document.getElementById("demo").innerHTML =
"El texto en este párrafo es " + miElemento.innerHTML;
</script>

</body>
</html>
```

Si el elemento es encontrado, el método nos devolverá el elemento como un objeto(en *miElemento*). Si el elemento no es encontrado *miElemento* contendrá *null*

Buscando elementos por el nombre de su etiqueta

Por ejemplo vamos a buscar los elementos <p>

```
<!DOCTYPE html>
<html>
<head>
```

```

<meta charset="utf-8">
</head>
<body>

<p>Hola Mundo</p>
<p>Este es un ejemplo del método <b>getElementsByName</b></p>

<p id="demo"></p>

<script>
var x = document.getElementsByTagName("p");
document.getElementById("demo").innerHTML =
'El primer párrafo (index 0) es: ' + x[0].innerHTML;
</script>

</body>
</html>

```

En el siguiente ejemplo buscaremos todos los elementos con el id="main" y entonces todos los elementos <p> dentro del "main":

```

<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
</head>

<body>

<p>Hola Mundo!</p>

<div id="main">
<p>El DOM es muy útil</p>
<p>Este ejemplo demuestra el método <b>getElementsByName</b></p>
</div>

<p id="demo"></p>

<script>

```

```

var x = document.getElementById("main");
var y = x.getElementsByTagName("p");
document.getElementById("demo").innerHTML =
'El primer párrafo (index 0) dentro de "main" es: ' + y[0].innerHTML;
</script>

</body>
</html>

```

Buscando elementos por el nombre de su clase

Para buscar un elemento HTML con un determinado nombre de clase usamos `getElementByClassName`

```

<!DOCTYPE html>
<html>
<body>

<p>Hello World!</p>

<p class="intro">DOM es muy útil.</p>
<p class="intro">Con este ejemplo demostramos el método <b>getElementsByTagName</b></p>

<p id="demo"></p>

<script>
var x = document.getElementsByClassName("intro");
document.getElementById("demo").innerHTML =
'El primer párrafo (index 0) con la clase="intro": ' + x[0].innerHTML;
</script>

</body>
</html>

```

Este método no funciona con **Internet Explorer 8** ni versiones anteriores.

Buscando elementos HTML por el selector CSS

Para buscar los elementos HTML que correspondan con un selector CSS en concreto usamos el método `querySelectorAll()`

En este ejemplo vamos a obtener una lista de todos los elementos `<p>` cuya clase sea "intro".


```

<!DOCTYPE html>
<html>
<body>

<p>Hola Mundo</p>

<p class="intro">El DOM es muy útil.</p>
<p class="intro">Este ejemplo demuestra el método <b>querySelectorAll</b>.</p>

<p id="demo"></p>

<script>
var x = document.querySelectorAll("p.intro");
document.getElementById("demo").innerHTML =
'El primer párrafo (index 0) con clase="intro": ' + x[0].innerHTML;
</script>

</body>
</html>

```

HTML DOM - Cambiando el HTML

el HTML DOM permite a JavaScript cambiar el contenido de los elementos HTML.

Cambiando la salida HTML

JavaScript puede crear dinámicamente contenido HTML

En JavaScript, *document.write()* puede ser usado para escribir directamente en la salida HTML
Ejemplo:

```

<!DOCTYPE html>
<html>
<body>

<script>
document.write(Date());
</script>

</body>
</html>

```

No hay que usar nunca `document.write()` después de que una página sea cargada o lo sobre escribirá todo

Cambiando contenido HTML

La manera más fácil de modificar el contenido de un elemento HTML es usar la propiedad **innerHTML**

Para cambiar el contenido de un elemento HTML usamos la sintaxis:

```
document.getElementById(id).innerHTML = new HTML
```

Ejemplo:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript puede cambiar HTML</h2>

<p id="p1">Hola Mundo</p>

<script>
document.getElementById("p1").innerHTML = "New text!";
</script>

</body>
</html>
```

Qué hemos hecho aquí:

1. El documento html contiene un elemento `<p>` con id p1
2. Usamos el HTML DOM para para obtener el elemento con id p1
3. JavaScript cambia el contenido

Cambiando el valor de un atributo

Para cambiar el valor de un atributo, usamos esta sintaxis

```
document.getElementById(id).attribute = new valor
```

Ejemplo:

```
<!DOCTYPE html>
<html>
<body>



<script>
document.getElementById("image").src = "landscape.jpg";
</script>

<p>esto era un gatito pero lo hemos cambiado por un paisaje</p>

</body>
</html>
```

Ejercicios hacer varios ejemplos con otros atributos

HTML DOM - Cambiando el CSS

El HTML DOM también le permite a JavaScript cambiar el estilo de los elementos HTML.

Cambiando el estilo

Para eso usamos la siguiente sintaxis:

```
document.getElementById(id).style.property = nuevo estilo
```

Veámoslo en un ejemplo:

```
<!DOCTYPE html>
<html>
<body>

<p id="p1">Hola Mundo!</p>
<p id="p2">Hola Mundo!</p>

<script>
document.getElementById("p2").style.color = "blue";
document.getElementById("p2").style.fontFamily = "Arial";
```

```
document.getElementById("p2").style.fontSize = "larger";  
</script>
```

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

Usando eventos

El HTML DOM permite ejecutar código cuando ocurre un evento.

Los eventos pueden producirse cuando

- Un elemento es pulsado
- La página ha sido cargada
- Un campo de entrada ha cambiado

Veremos más eventos en el siguiente capítulo

De momento veremos un ejemplo de como cambiar un elemento al pulsar un botón.

Ejemplo:

```
<!DOCTYPE html>  
<html>  
<body>  
  
<h1 id="id1">Si me miras me pongo rojito</h1>  
  
<button type="button"  
onclick="document.getElementById('id1').style.color = 'red'">  
Pulsame!</button>  
  
</body>  
</html>
```

Otro ejemplo sería como podemos alterar la visibilidad de un elemento HTML

```
<!DOCTYPE html>  
<html>  
<body>
```

```

<p id="p1"> Hola soy un texto molón que aparece y desaparece </p>

<input type="button" value="Esconder texto" onclick="document.getElementById('p1').style.visibility=
<input type="button" value="Mostrar texto" onclick="document.getElementById('p1').style.visibility=

</body>
</html>

```

Más adelante veremos más propiedades que se pueden cambiar

Ejercicios

Animaciones con HTML DOM

Vamos a aprender los principios de como hacer animaciones con JavaScript

Una página básica

Para hacer la demostración crearemos una página sencilla

```

<!DOCTYPE html>
<html>
<body>
<h1>Mi primera animación con JavaScript</h1>
<div id="animate">Mi animación irá aquí</div>
</body>
</html>

```

Creamos un contenedor para la animación

Todas las animaciones deberán estar relativas a un elemento contenedor

```

<div id="container">
<div id="animate">
</div>

```

Estilos

El contenedor debe ser creado con un style="position: relative". El elemento de la animación debe ser con creado con style="position: absolute"

Ejemplo:

```

#container {
    width: 400px;
    height: 400px;
    position: relative;
    background: yellow;
}
#animate {
    width: 50px;
    height: 50px;
    position: absolute;
    background: red;
}

```

Código para la animación

La animación se realiza programando cambios graduales en el estilo del elemento. Los cambios serán llamados por un temporizador. Cuando los intervalos son pequeños, la animación parece continua.

El código básico es:

```

var id = setInterval(frame, 5);

function frame() {
    if (/* test for finished */) {
        clearInterval(id);
    } else {
        /* code to change the element style */
    }
}

```

Creando la animación usando JavaScript

```

function myMove() {
    var elem = document.getElementById("animate");
    var pos = 0;
    var id = setInterval(frame, 5);
    function frame() {
        if (pos == 350) {
            clearInterval(id);
        } else {
            pos++;
            elem.style.top = pos + 'px';
        }
    }
}

```

```

        elem.style.left = pos + 'px';
    }
}

```

Todo junto finalmente

```

<!DOCTYPE html>
<html>
<style>
#container {
    width: 400px;
    height: 400px;
    position: relative;
    background: yellow;
}
#animate {
    width: 50px;
    height: 50px;
    position: absolute;
    background-color: red;
}
</style>
<body>

<p>
<button onclick="myMove()">Pulsa</button>
</p>

<div id="container">
<div id="animate"></div>
</div>

<script>
function myMove() {
    var elem = document.getElementById("animate");
    var pos = 0;
    var id = setInterval(frame, 5);
    function frame() {
        if (pos == 350) {

```

```

        clearInterval(id);
    } else {
        pos++;
        elem.style.top = pos + 'px';
        elem.style.left = pos + 'px';
    }
}
}
</script>

</body>
</html>

```

Eventos en el HTML DOM

El HTML DOM permite a JavaScript reaccionar a los eventos HTML

Reaccionando a los eventos

JavaScript puede ser ejecutado cuando un evento ocurre, como el pulsar un elemento HTML. Para ejecutar código cuando un usuario pulsa un elemento, añadimos código JavaScript a un atributo de evento html.

Ejemplo:

onclick=JavaScript

Ejemplos de eventos html son:

- Cuando un usuario pulsa con el ratón
- Cuando una página es cargada
- Cuando una imagen es cargada
- Cuando con el ratón se pasa sobre un elemento
- Cuando un cuadro de entrada es cambiado
- Cuando un formulario html es enviado
- Cuando un usuario pulsa una tecla

Veamos un ejemplo, donde el contenido de una cabecera es cambiado cuando pulsamos un botón.


```

<!DOCTYPE html>
<html>
<body>

<h1 onclick="this.innerHTML='Holaaaaa!'">Pulsa este texto!</h1>

</body>
</html>

```

En este ejemplo, una función es llamada cuando ocurre el evento

```

<!DOCTYPE html>
<html>
<body>

<h1 onclick="changeText(this)">Pulsa este texto!</h1>

<script>
function changeText(id) {
    id.innerHTML = "Holaaaaa!";
}
</script>

</body>
</html>

```

Atributos de eventos

Para asignar eventos a elementos html podemos usar atributos de eventos.

Ejemplo:

```

<button onclick="displayDate()">Púlsame</button>

<!DOCTYPE html>
<html>
<body>

<p>Este botón da la hora.</p>

<button onclick="displayDate()">Qué hora es?</button>

```

```

<script>
function displayDate() {
    document.getElementById("demo").innerHTML = Date();
}
</script>

<p id="demo"></p>

</body>
</html>

```

Asignando eventos usando el HTML DOM

El HTML DOM permite asignar eventos a los elementos html usando JavaScript:

Ejemplo:

```

document.getElementById("miBtn").onclick = displayDate;

<!DOCTYPE html>
<html>
<body>

<p>Si pulsamos el botón ejecutamos la función displayDate().</p>

<button id="myBtn">Pulsame</button>

<p id="demo"></p>

<script>
document.getElementById("miBtn").onclick = displayDate;

function displayDate() {
    document.getElementById("demo").innerHTML = Date();
}
</script>

</body>
</html>

```

En el ejemplo de arriba, una función llamada *displayDate* s asignada a un elemento html con el id="miBtn".

Los eventos onload y onunload

El onload y onunload son usado para saber si el usuario entra o deja la página. El onload puede ser usado para comprobar el navegador con el que están visitando nuestra página y ofrecer así el contenido adaptado a ese navegador o también para lidiar con las cookies.

```
<!DOCTYPE html>
<html>
<body onload="checkCookies()">

<p id="demo"></p>

<script>
function checkCookies() {
    var text = "";
    if (navigator.cookieEnabled == true) {
        text = "Las cookies están habilitadas.";
    } else {
        text = "Las cookies no están habilitadas.";
    }
    document.getElementById("demo").innerHTML = text;
}
</script>

</body>
</html>
```

Evento onchange

Es usado en combinación de validadores de campos de entrada. En el siguiente ejemplo la función *upperCase()* será llamada cuando el usuario cambie el contenido de un campo de entrada.

Ejemplo:

```
<!DOCTYPE html>
<html>
<head>
<script>
function myFunction() {
    var x = document.getElementById("fnombre");
    x.value = x.value.toUpperCase();
}

```

```
</script>
</head>
<body>
```

Escribe tu nombre: `<input type="text" id="fnombre" onchange="myFunction()">`

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

Evento onmouseover y onmouseout

Esos eventos pueden ser usados para cambiar elementos HTML cuando el ratón pasa por encima de esos elementos.

```
<!DOCTYPE html>
<html>
<body>

<div onmouseover="mOver(this)" onmouseout="mOut(this)"
style="background-color:#D94A38;width:120px;height:20px;padding:40px;">
Tócame</div>

<script>
function mOver(obj) {
    obj.innerHTML = "Tócame";
}

function mOut(obj) {
    obj.innerHTML = "Gracias, sigue tocando @-@";
}
</script>

</body>
</html>
```

El evento onmousedown, onmouseup y onclick

Estos eventos son todos parte de mouse-click.

Veamos un ejemplo:

```
<!DOCTYPE html>
<html>
```

```

<body>

<div onmousedown="mDown(this)" onmouseup="mUp(this)"
style="background-color:#D94A38;width:90px;height:20px;padding:40px;">
Pulsame</div>

<script>
function mDown(obj) {
    obj.style.backgroundColor = "#1ec5e5";
    obj.innerHTML = "Pulsado";
}

function mUp(obj) {
    obj.style.backgroundColor="#D94A38";
    obj.innerHTML="Gracias";
}
</script>

</body>
</html>

```

Otros ejemplos:

Con onload:

```

<!DOCTYPE html>
<html>
<head>

<script>
function mymessage() {
    alert("This message was triggered from the onload event");
}
</script>
</head>

<body onload="mymessage()">
</body>

</html>

```

Con onfocus:

```

<!DOCTYPE html>
<html>
<head>
<script>
function myFunction(x) {
    x.style.background = "yellow";
}
</script>
</head>
<body>

```

Escribe tu nombre: <input type="text" onfocus="myFunction(this)">

```

<p>Cuando la caja de texto obtiene el foco, ésta cambia el fondo de color</p>
</body>
</html>

```

Con onmouseover:

```

<!DOCTYPE html>
<html>
<body>

<h1 onmouseover="style.color='red'"
onmouseout="style.color='black'">
Pasa el ratón por el texto</h1>

</body>
</html>

```

Colecciones de Objetos

Cuando usamos el método getElementById éste devuelve una **colección de objetos HTML** Una colección de objetos es como un Array. Así que para luego acceder a ellos usamos la misma forma que con la que accedemos a los elementos de los Arrays.

Ejemplo:

```

<!DOCTYPE html>
<html>
<body>

```

```

<h2>Colecciones de objetos</h2>

<p>Hola Mundo!</p>

<p>Adios Mundo cruel!</p>

<p id="demo"></p>

<script>
var myCollection = document.getElementsByTagName("p");
document.getElementById("demo").innerHTML =
"El segundo objeto de la colección es: " +
myCollection[1].innerHTML;
</script>

</body>
</html>

```

Funciones en JavaScript

Para crear una función en JavaScript usamos la palabra clave **function**

Declarando funciones

Las funciones son declaradas con la siguiente sintaxis:

```

function nombreFuncion(parámetros){
    código que va a ser ejecutado
}

```

Al declarar una función no será ejecutada inmediatamente sino que será guardada para ser ejecutada posteriormente.

Ejemplo:

```

function multiplicar(a, b){
    return a * b;
}

```

```

<!DOCTYPE html>
<html>

```

```

<body>

<h2>Funciones en JavaScript</h2>

<p id="demo"></p>

<script>
function multiplicar(a, b) {
    return a * b;
}
document.getElementById("demo").innerHTML = multiplicar(4, 3);
</script>

</body>
</html>

```

Expresiones

En JavaScript una función también puede ser definida usando una expresión y además ser guardada en una variable.

```

<!DOCTYPE html>
<html>
<body>

<p id="demo"></p>

<script>
var x = function (a, b) {return a * b};
document.getElementById("demo").innerHTML = x(4, 3);
</script>

</body>
</html>

```

Estas son también llamadas **funciones anónimas** (sin nombre). Para ser invocadas usaremos el nombre de la variable.

El constructor function()

Hemos visto como cómo definir funciones con la palabra **function**.

Otra forma de hacerlo es con el constructor **function()**

Ejemplo:

```
<!DOCTYPE html>
<html>
<body>

<p id="demo"></p>

<script>
var myFunction = new Function("a", "b", "return a * b");
document.getElementById("demo").innerHTML = myFunction(4, 3);
</script>

</body>
</html>
```

Las funciones pueden ser llamadas antes de declaradas

```
myFunction(5);
```

```
function myFunction(y) {
    return y * y;
}
```

Objetos en JavaScript

En JavaScript todo es un objeto:

- Booleans pueden ser objetos si se definen con la palabra clave new
- Los números pueden ser objetos si se definen con la palabra clave new
- Las cadenas de texto pueden ser objetos si se definen con la palabra clave new
- Las fechas son objetos
- Las expresiones regulares son objetos

- Las funciones son objetos
- Y los objetos también son objetos

Primitivos en JavaScript

Un **valor primitivo** es un valor que no tiene propiedades o métodos. Un tipo de dato primitivo es aquel que tiene un valor primitivo. JavaScript define 5 tipos de datos primitivos.

- Cadenas de texto
- Números
- booleanos
- null
- indefinidos

Los objetos son variables conteniendo variables

Las variables en JavaScript pueden contener valores simples

```
var persona = "Paquito Chocolatero";
```

Los objetos son variables también. Pero los objetos pueden contener muchos valores

```
var persona = {nombre:"Paquito", apellido:"Chocolatero", edad:100}
```

Propiedades del objeto

Los nombres de los valores, en los objetos de JavaScript son llamados **propiedades**.

| Propiedades | Valor |
|-------------|-------------|
| nombre | Paquito |
| apellidos | Chocolatero |
| edad | 100 |

Las propiedades de un objeto pueden ser valores primitivos, otros objetos o funciones.

Métodos del objeto

Los métodos son **acciones** que pueden ser desarrolladas en los objetos. Un **método** es una propiedad del objeto que contiene una definición de función.

| Propiedades | Valor |
|----------------|---|
| nombre | Paquito |
| apellido | Chocolatero |
| edad | 100 |
| nombreCompleto | function(){return this.nombre + " " + this.apellido;} |

Creando un objeto en JavaScript

Con JavaScript, podemos definir y crear nuestros propios objetos.

Hay diferentes formas de crear un nuevo objeto:

- Definir y crear un simple objeto, usando un objeto literal
- Definir y crear un simple objeto, con la palabra clave new
- Definir un objeto constructor y entonces crear objetos del tipo del constructor

Usando un Objeto Literal

Es la forma más fácil de crear un objeto en JavaScript. Usando un objeto literal, en una única declaración creamos y definimos un objeto.

Un objeto literal es una lista de pares nombre:valor entre llaves.

Ejemplo:

```
var persona = {nombre:"Paquito", apellidos:"Chocolatero", edad:100};
```

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<p>Creando un objeto en Java.</p>
```

```
<p id="demo"></p>
```

```
<script>
```

```
var person = {nombre:"Paquito", apellido:"Chocolatero", edad:100};
```

```
document.getElementById("demo").innerHTML =
person.nombre + " tiene " + persona.edad + " años.";
</script>

</body>
</html>
```

Usando la palabra new

En el siguiente ejemplo también creamos un nuevo objeto con cuatro propiedades:

```
var persona = new Object();
persona.nombre = "Paquito";
persona.apellido = "Chocolatero";
persona.edad = 100;
```

Usando un Objeto Constructor

Lo que hemos visto es muy limitado porque creamos un único objeto. A veces nosotros queremos tener un "objeto tipo" que pueda ser usado para crear muchos objetos de ese tipo. Para ello vamos a usar una función constructora de objetos.

Ejemplo:

```
function persona(nombre, apellido, edad){
    this.nombre = nombre;
    this.apellido = apellido;
    this.edad = edad;
}

var miPadre = new persona("Eusebio", "Martínez", 55);
var miMadre = new persona("Maria", "García", 50);
```

Hagamos la prueba:

```
<!DOCTYPE html>
<html>
<body>

<p id="demo"></p>

<script>
```

```
function persona(nombre, apellido, edad){
    this.nombre = nombre;
    this.apellido = apellido;
    this.edad = edad;
}

var miPadre = new persona("Eusebio", "Martínez", 55);
var miMadre = new persona("Maria", "García", 50);

document.getElementById("demo").innerHTML =
"Mi padre tiene " + miPadre.edad "años de edad" + ". Mi Madre tiene " + miMadre.edad "años de edad"
</script>

</body>
</html>
```

Propiedades en Objetos JavaScript

Las propiedades es lo más importante de un objeto. Son valores asociados al objeto.

Las propiedades pueden ser cambiadas, añadidas y borradas, pero algunos son de solo lectura.

Accediendo a las propiedades

Hay varias formas de acceder a una propiedad

nombreObjeto.propiedad *//persona.edad*

ó

nombreObjeto["propiedad"] *//persona["edad"]*

ó

nombreObjeto[expresión] *// x = "edad"; persona[x]*

Ejemplo:

```
<!DOCTYPE html>
<html>
<body>
```

```
<p>Veamos la forma de acceder a las propiedades de un objeto</p>
```

```

<p id="demo"></p>

<script>
var persona = {
    nombre:"Paquito",
    apellido:"Chocolatero",
    edad:100,
};
document.getElementById("demo").innerHTML =
persona.nombre + " tiene " + persona.edad + " años de edad.";
</script>

</body>
</html>

```

Añadiendo nuevas propiedades

Podemos añadir una nueva propiedad a un objeto que ya exista simplemente dándole un valor.

Ejemplo:

```

persona.nacionalidad = "Española";

```

Veamos:

```

<!DOCTYPE html>
<html>
<body>

<p>Veamos la forma de acceder a las propiedades de un objeto</p>

<p id="demo"></p>

<script>
var persona = {
    nombre:"Paquito",
    apellido:"Chocolatero",
    edad:100,
};

```

```

persona.nacionalidad = "Española";
document.getElementById("demo").innerHTML =
persona.nombre + " tiene " + persona.edad + " años de edad." + "y tiene nacionalidad " + persona.na
</script>

</body>
</html>

```

Borrando una propiedad

Para ello usaremos la palabra clave **delete**

```
delete persona.edad;
```

Veamos como funciona:

```

<html>
<body>

<p>Veamos la forma de acceder a las propiedades de un objeto</p>

<p id="demo"></p>

<script>
var persona = {
    nombre:"Paquito",
    apellido:"Chocolatero",
    edad:100,
};
delete persona.edad;
document.getElementById("demo").innerHTML =
persona.nombre + " tiene " + persona.edad + " años de edad.";
</script>

</body>
</html>

```

Delete borrará tanto el valor como la propiedad.

Métodos en JavaScript

Los métodos son las acciones que pueden ejecutarse en los objetos.

Accediendo a los métodos

Creamos un método de objeto con la siguiente sintaxis:

```
nombreMetodo : function(){ líneas de código }
```

Y accedemos a ese método

```
nombreObjeto.nombreMetodo()
```

Añadiendo nuevos métodos

Para añadir nuevos métodos lo tenemos que hacer en el constructor.

Ejemplo:

```
<!DOCTYPE html>
<html>
<body>

<p id="demo"></p>

<script>
function persona(nombre,apellido,edad) {
    this.nombre = nombre;
    this.apellido = apellido;
    this.edad = edad;
    this.cambiarNombre = function (nombre) {
        this.apellido = name;
    }
}
var miMadre = new persona("María","García",50);
miMadre.cambiarNombre("Cano");
document.getElementById("demo").innerHTML =
"El apellido de mi madre es " + miMadre.apellido;
</script>

</body>
</html>
```