


# JAVASCRIPT

A solid yellow square that serves as a background for the 'JS' text.

# JS

# JAVASCRIPT

## LIVESCRIPT - JAVASCRIPT

Es lenguaje de programación dinámica.

## CLIENT-SIDE

### **LIMITACIONES INICIALES**

No permite lectura y ni escritura de archivos del servidor

No se puede usar en estructuras de RED.

No tiene estructuras multihilo o múltiples procesos simultáneos.

# JAVASCRIPT

## Tipos primitivos

Javascript tiene seis tipos primitivos:

- Sin definir (undefined)
- Nulo (null)
- Lógicos (boolean)
- Numérico (number)
- Cadena (string)
- Símbolo (symbol) Lo veremos en ES6

Todos los demás tipos son objetos (Object): Array, Date, Promise y todos los demás tipos son objetos.

# JAVASCRIPT

## OPERADORES

### **Asignación**

El operador de asignación es el más utilizado y el más sencillo. Este operador se utiliza para guardar un valor específico en una variable. El símbolo utilizado es = (no confundir con el operador == que se verá más adelante):

```
var numero = 3;
```

# JAVASCRIPT

## OPERADORES

### Incremento y decremento

Estos dos operadores solamente son válidos para las variables numéricas y se utilizan para incrementar o decrementar en una unidad el valor de una variable.

Ejemplo:

```
var numero = 5;
```

Incremento -> `numero++` es equivalente `numero = numero + 1` (`alert(numero)` -> 6)

Decremento -> `numero--` es equivalente a `numero = numero - 1` (`alert(numero)` -> 4)

# JAVASCRIPT

## OPERADORES

### Aritméticos o matemáticos

JavaScript permite realizar manipulaciones matemáticas sobre el valor de las variables numéricas. Los operadores definidos son: suma (+), resta (-), multiplicación (\*) y división (/). Ejemplo:

```
var numero1 = 10;  
var numero2 = 5;
```

```
resultado = numero1 / numero2;    // resultado = 2  
resultado = 3 + numero1;          // resultado = 13  
resultado = numero2 - 4;          // resultado = 1  
resultado = numero1 * numero 2;   // resultado = 50
```

Además de los cuatro operadores básicos, JavaScript define otro operador matemático que no es sencillo de entender cuando se estudia por primera vez, pero que es muy útil en algunas ocasiones. Se trata del operador "*módulo*", que calcula el resto de la división entera de dos números.

```
var numero1 = 10;  
var numero2 = 5;  
resultado = numero1 % numero2;    // resultado = 0
```

# JAVASCRIPT

## OPERADORES RELACIONALES O DE COMPARACIÓN

Los operadores relacionales definidos por JavaScript son idénticos a los que definen las matemáticas: mayor que (>), menor que (<), mayor o igual (>=), menor o igual (<=), igual que (==) y distinto de (!=). Su resultado es true o false.

```
var numero1 = 3;
var numero2 = 5;

resultado = numero1 > numero2; // resultado = false
resultado = numero1 < numero2; // resultado = true

numero1 = 5;
numero2 = 5;

resultado = numero1 >= numero2; // resultado = true
resultado = numero1 <= numero2; // resultado = true
resultado = numero1 == numero2; // resultado = true
resultado = numero1 != numero2; // resultado = false
```

# JAVASCRIPT

## Operadores Lógicos

Los operadores lógicos son imprescindibles para realizar aplicaciones complejas, ya que se utilizan para tomar decisiones sobre las instrucciones que debería ejecutar el programa en función de ciertas condiciones.

El resultado de cualquier operación que utilice operadores lógicos siempre es un valor lógico o *booleano*.

### Negación

Uno de los operadores lógicos más utilizados es el de la negación. Se utiliza para obtener el valor contrario al valor de la variable:

```
var visible = true;  
alert(!visible); // Muestra "false" y no "true"
```

La negación lógica se obtiene prefijando el símbolo ! al identificador de la variable. El funcionamiento de este operador se resume en la siguiente tabla:

variable	!variable
true	false
false	true



# JAVASCRIPT

## Operadores Lógicos

Los operadores lógicos son imprescindibles para realizar aplicaciones complejas, ya que se utilizan para tomar decisiones sobre las instrucciones que debería ejecutar el programa en función de ciertas condiciones.

El resultado de cualquier operación que utilice operadores lógicos siempre es un valor lógico o *booleano*.

### AND (&&)

La operación lógica AND obtiene su resultado combinando dos valores booleanos. Su resultado solamente es true si los dos operandos son true:

variable1	variable2	variable1 && variable2
true	true	true
true	false	false
false	true	false
false	false	false

# JAVASCRIPT

## Operadores Lógicos

Los operadores lógicos son imprescindibles para realizar aplicaciones complejas, ya que se utilizan para tomar decisiones sobre las instrucciones que debería ejecutar el programa en función de ciertas condiciones.

El resultado de cualquier operación que utilice operadores lógicos siempre es un valor lógico o *booleano*.

### OR ( || )

La operación lógica OR obtiene su resultado combinando dos valores booleanos. Su resultado es true si alguno de los dos operandos es true

variable1	variable2	variable1    variable2
true	true	true
true	false	true
false	true	true
false	false	false

# JAVASCRIPT

## Operadores Concatenación

Se dan en el manejo de cadenas de caracteres o string. Permiten unir el valor de una cadena de caracteres que está almacenada en una variable a otro nuevo y así no perder el valor que ya estaba almacenado.

EJEMPLO:

```
var nombre = "Juan";  
var apellido = "Pérez";
```

**var nombreCompleto = nombre + apellido // resultado "JuanPérez"**

Debemos recordar que el espacio también es un caracteres y lo tenemos que contemplar en nuestros desarrollos.

**var nombreCompleto = nombre + " " + apellido // resultado "Juan Pérez"**

También podemos sobrescribir las variables usando la concatenación

**var nombre = nombre + " " + apellido // nombre = "Juan Pérez" es equivalente a nombre += " "+apellido**

## DOM

La creación del *Document Object Model* o **DOM** es una de las innovaciones que más ha influido en el desarrollo de las páginas web dinámicas y de las aplicaciones web más complejas.

DOM permite a los programadores web acceder y manipular las páginas HTML como si fueran documentos XML. De hecho, DOM se diseñó originalmente para manipular de forma sencilla los documentos XML.

A pesar de sus orígenes, DOM se ha convertido en una utilidad disponible para la mayoría de lenguajes de programación (Java, PHP, JavaScript) y cuyas únicas diferencias se encuentran en la forma de implementarlo.

## DOM

La especificación completa de DOM define muchos tipos de nodos, aunque las páginas HTML habituales se pueden manipular manejando solamente cuatro o cinco tipos de nodos:

- `Document`, nodo raíz del que derivan todos los demás nodos del árbol.
- `Element`, representa cada una de las etiquetas HTML. Se trata del único nodo que puede contener atributos y el único del que pueden derivar otros nodos.
- `Attr`, se define un nodo de este tipo para representar cada uno de los atributos de las etiquetas HTML, es decir, uno por cada par `atributo=valor`.
- `Text`, nodo que contiene el texto encerrado por una etiqueta HTML.
- `Comment`, representa los comentarios incluidos en la página HTML.

## DOM

La función `getElementsByTagName(nombreEtiqueta)` obtiene todos los elementos de la página HTML cuya etiqueta sea igual que el parámetro que se le pasa a la función.

El siguiente ejemplo muestra cómo obtener todos los párrafos de una página XHTML:

```
var parrafos = document.getElementsByTagName("p");
```

El valor que se indica delante del nombre de la función (en este caso, `document`) es el nodo a partir del cual se realiza la búsqueda de los elementos. En este caso, como se quieren obtener todos los párrafos de la página, se utiliza el valor `document` como punto de partida de la búsqueda.

## DOM

El valor que devuelve la función es un array con todos los nodos que cumplen la condición de que su etiqueta coincide con el parámetro proporcionado. El valor devuelto es un array de nodos DOM, no un array de cadenas de texto o un array de objetos normales. Por lo tanto, se debe procesar cada valor del array de la esta forma

```
var primerParrafo = parrafos[0];
```

De la misma forma, se podrían recorrer todos los párrafos de la página con el siguiente código:

```
for(var i=0; i<parrafos.length; i++) {  
    var parrafo = parrafos[i];  
}
```

## DOM

La función `getElementsByTagName()` se puede aplicar de forma recursiva sobre cada uno de los nodos devueltos por la función. En el siguiente ejemplo, se obtienen todos los enlaces del primer párrafo de la página:

```
var parrafos = document.getElementsByTagName("p");  
var primerParrafo = parrafos[0];  
var enlaces = primerParrafo.getElementsByTagName("a");
```



## DOM

La función `getElementByName()` es similar a la anterior, pero en este caso se buscan los elementos cuyo atributo `name` sea igual al parámetro proporcionado. En el siguiente ejemplo, se obtiene directamente el único párrafo con el nombre indicado:

```
var parrafoEspecial = document.getElementById("especial");
```

```
<p name="prueba">...</p>
```

```
<p name="especial">...</p>
```

```
<p>...</p>
```

## DOM

Normalmente el atributo `name` es único para los elementos HTML que lo definen, por lo que es un método muy práctico para acceder directamente al nodo deseado. En el caso de los elementos HTML *radiobutton*, el atributo `name` es común a todos los *radiobutton* que están relacionados, por lo que la función devuelve una colección de elementos.

Internet Explorer 6.0 no implementa de forma correcta esta función, ya que sólo la tiene en cuenta para los elementos de tipo `<input>` y `<img>`. Además, también tiene en consideración los elementos cuyo atributo `id` sea igual al parámetro de la función.

## DOM

La función `getElementById()` es la más utilizada cuando se desarrollan aplicaciones web dinámicas. Se trata de la función preferida para acceder directamente a un nodo y poder leer o modificar sus propiedades.

La función `getElementById()` devuelve el elemento XHTML cuyo atributo `id` coincide con el parámetro indicado en la función. Como el atributo `id` debe ser único para cada elemento de una misma página, la función devuelve únicamente el nodo deseado.

```
var cabecera = document.getElementById("cabecera");  
  
<div id="cabecera">  
  <a href="/" id="logo">...</a>  
</div>
```

## DOM

Como se ha visto, un elemento HTML sencillo, como por ejemplo un párrafo, genera dos nodos: el primer nodo es de tipo `Element` y representa la etiqueta `<p>` y el segundo nodo es de tipo `Text` y representa el contenido textual de la etiqueta `<p>`.

Por este motivo, crear y añadir a la página un nuevo elemento HTML sencillo consta de cuatro pasos diferentes:

1. Creación de un nodo de tipo `Element` que represente al elemento.
2. Creación de un nodo de tipo `Text` que represente el contenido del elemento.
3. Añadir el nodo `Text` como nodo hijo del nodo `Element`.
4. Añadir el nodo `Element` a la página, en forma de nodo hijo del nodo correspondiente al lugar en el que se quiere insertar el elemento.

## DOM

De este modo, si se quiere añadir un párrafo simple al final de una página HTML, es necesario incluir el siguiente código JavaScript:

```
// Crear nodo de tipo Element
var parrafo = document.createElement("p");
// Crear nodo de tipo Text
var contenido = document.createTextNode("Hola Mundo!");
// Añadir el nodo Text como hijo del nodo Element
parrafo.appendChild(contenido);
// Añadir el nodo Element como hijo de la pagina
document.body.appendChild(parrafo);
```

## DOM

El proceso de creación de nuevos nodos puede llegar a ser tedioso, ya que implica la utilización de tres funciones DOM:

- `createElement(etiqueta)`: crea un nodo de tipo `Element` que representa al elemento XHTML cuya etiqueta se pasa como parámetro.
- `createTextNode(contenido)`: crea un nodo de tipo `Text` que almacena el contenido textual de los elementos HTML.
- `nodoPadre.appendChild(nodoHijo)`: añade un nodo como hijo de otro nodo. Se debe utilizar al menos dos veces con los nodos habituales: en primer lugar se añade el nodo `Text` como hijo del nodo `Element` y a continuación se añade el nodo `Element` como hijo de algún nodo de la página.

## DOM

Afortunadamente, eliminar un nodo del árbol DOM de la página es mucho más sencillo que añadirlo. En este caso, solamente es necesario utilizar la función `removeChild()`:

```
var parrafo = document.getElementById("provisional");  
parrafo.parentNode.removeChild(parrafo);
```

```
<p id="provisional">...</p>
```

## DOM

La función `removeChild()` requiere como parámetro el nodo que se va a eliminar. Además, esta función debe ser invocada desde el elemento padre de ese nodo que se quiere eliminar. La forma más segura y rápida de acceder al nodo padre de un elemento es mediante la propiedad `nodoHijo.parentNode`.

Así, para eliminar un nodo de una página HTML se invoca a la función `removeChild()` desde el valor `parentNode` del nodo que se quiere eliminar. Cuando se elimina un nodo, también se eliminan automáticamente todos los nodos hijos que tenga, por lo que no es necesario borrar manualmente cada nodo hijo.



## DOM

Una vez que se ha accedido a un nodo, el siguiente paso natural consiste en acceder y/o modificar sus atributos y propiedades. Mediante DOM, es posible acceder de forma sencilla a todos los atributos XHTML y todas las propiedades CSS de cualquier elemento de la página.

Los atributos XHTML de los elementos de la página se transforman automáticamente en propiedades de los nodos. Para acceder a su valor, simplemente se indica el nombre del atributo XHTML detrás del nombre del nodo.

## DOM

El siguiente ejemplo obtiene de forma directa la dirección a la que enlaza el enlace:

```
var enlace = document.getElementById("enlace");  
alert(enlace.href); // muestra http://www...com
```

```
<a id="enlace" href="http://www...com">Enlace</a>
```

## DOM

En el ejemplo anterior, se obtiene el nodo DOM que representa el enlace mediante la función `document.getElementById()`. A continuación, se obtiene el atributo `href` del enlace mediante `enlace.href`. Para obtener por ejemplo el atributo `id`, se utilizaría `enlace.id`.

Las propiedades CSS no son tan fáciles de obtener como los atributos XHTML. Para obtener el valor de cualquier propiedad CSS del nodo, se debe utilizar el atributo `style`. El siguiente ejemplo obtiene el valor de la propiedad `margin` de la imagen:

## DOM

```
var imagen = document.getElementById("imagen");  
alert(imagen.style.margin);
```

```

```

Si el nombre de una propiedad CSS es compuesto, se accede a su valor modificando ligeramente su nombre:

## DOM

```
var parrafo = document.getElementById("parrafo");  
alert(parrafo.style.fontWeight); // muestra "bold"  
<p id="parrafo" style="font-weight: bold;">...</p>
```

La transformación del nombre de las propiedades CSS compuestas consiste en eliminar todos los guiones medios (-) y escribir en mayúscula la letra siguiente a cada guión medio. A continuación se muestran algunos ejemplos:

## DOM

- `font-weight` se transforma en `fontWeight`
- `line-height` se transforma en `lineHeight`
- `border-top-style` se transforma en `borderTopStyle`
- `list-style-image` se transforma en `listStyleImage`

El único atributo HTML que no tiene el mismo nombre en HTML y en las propiedades DOM es el atributo `class`. Como la palabra `class` está reservada por JavaScript, no es posible utilizarla para acceder al atributo `class` del elemento HTML. En su lugar, DOM utiliza el nombre `className` para acceder al atributo `class` de HTML:

```
var parrafo = document.getElementById("parrafo");
```

```
alert(parrafo.class); // muestra "undefined"
```

```
alert(parrafo.className); // muestra "normal"
```

```
<p id="parrafo" class="normal">...</p>
```

## DOM

- `font-weight` se transforma en `fontWeight`
- `line-height` se transforma en `lineHeight`
- `border-top-style` se transforma en `borderTopStyle`
- `list-style-image` se transforma en `listStyleImage`

El único atributo HTML que no tiene el mismo nombre en HTML y en las propiedades DOM es el atributo `class`. Como la palabra `class` está reservada por JavaScript, no es posible utilizarla para acceder al atributo `class` del elemento HTML. En su lugar, DOM utiliza el nombre `className` para acceder al atributo `class` de HTML:

```
var parrafo = document.getElementById("parrafo");
```

```
alert(parrafo.class); // muestra "undefined"
```

```
alert(parrafo.className); // muestra "normal"
```

```
<p id="parrafo" class="normal">...</p>
```

## DOM

A partir de la página web proporcionada y utilizando las funciones DOM, mostrar por pantalla la siguiente información:

1. Número de enlaces de la página
2. Dirección a la que enlaza el penúltimo enlace
3. Número de enlaces que enlazan a <http://prueba>
4. Número de enlaces del tercer párrafo



## EVENTOS

Hasta ahora, todas las aplicaciones y scripts que se han creado tienen algo en común: se ejecutan desde la primera instrucción hasta la última de forma secuencial. Gracias a las estructuras de control de flujo (`if`, `for`, `while`) es posible modificar ligeramente este comportamiento y repetir algunos trozos del script y saltarse otros trozos en función de algunas condiciones.

Este tipo de aplicaciones son poco útiles, ya que no interactúan con los usuarios y no pueden responder a los diferentes *eventos* que se producen durante la ejecución de una aplicación.

Afortunadamente, las aplicaciones web creadas con el lenguaje JavaScript pueden utilizar el modelo de *programación basada en eventos*.

## EVENTOS

En este tipo de programación, los scripts se dedican a esperar a que el usuario *"haga algo"* (que pulse una tecla, que mueva el ratón, que cierre la ventana del navegador). A continuación, el script responde a la acción del usuario normalmente procesando esa información y generando un resultado.

Los eventos hacen posible que los usuarios transmitan información a los programas. JavaScript define numerosos eventos que permiten una interacción completa entre el usuario y las páginas/aplicaciones web. La pulsación de una tecla constituye un evento, así como pinchar o mover el ratón, seleccionar un elemento de un formulario, redimensionar la ventana del navegador, etc.

JavaScript permite asignar una función a cada uno de los eventos. De esta forma, cuando se produce cualquier evento, JavaScript ejecuta su función asociada. Este tipo de funciones se denominan *"event handlers"* en inglés y suelen traducirse por *"manejadores de eventos"*

Evento	Descripción	Elementos para los que está definido
<code>onblur</code>	Deseleccionar el elemento	<code>&lt;button&gt;</code> , <code>&lt;input&gt;</code> , <code>&lt;label&gt;</code> , <code>&lt;select&gt;</code> , <code>&lt;textarea&gt;</code> , <code>&lt;body&gt;</code>
<code>onchange</code>	Deseleccionar un elemento que se ha modificado	<code>&lt;input&gt;</code> , <code>&lt;select&gt;</code> , <code>&lt;textarea&gt;</code>
<code>onclick</code>	Pinchar y soltar el ratón	Todos los elementos
<code>ondblclick</code>	Pinchar dos veces seguidas con el ratón	Todos los elementos

<code>onfocus</code>	Seleccionar un elemento	<code>&lt;button&gt;</code> , <code>&lt;input&gt;</code> , <code>&lt;label&gt;</code> , <code>&lt;select&gt;</code> , <code>&lt;textarea&gt;</code> , <code>&lt;body&gt;</code>
<code>onkeydown</code>	Pulsar una tecla (sin soltar)	Elementos de formulario y <code>&lt;body&gt;</code>
<code>onkeypress</code>	Pulsar una tecla	Elementos de formulario y <code>&lt;body&gt;</code>
<code>onkeyup</code>	Soltar una tecla pulsada	Elementos de formulario y <code>&lt;body&gt;</code>
<code>onload</code>	La página se ha cargado completamente	<code>&lt;body&gt;</code>

<code>onmousemove</code>	Mover el ratón	Todos los elementos
<code>onmouseout</code>	El ratón " <i>sale</i> " del elemento (pasa por encima de otro elemento)	Todos los elementos
<code>onmouseover</code>	El ratón " <i>entra</i> " en el elemento (pasa por encima del elemento)	Todos los elementos
<code>onmouseup</code>	Soltar el botón que estaba pulsado en el ratón	Todos los elementos
<code>onreset</code>	Inicializar el formulario (borrar todos sus datos)	<code>&lt;form&gt;</code>

<code>onresize</code>	Se ha modificado el tamaño de la ventana del navegador	<code>&lt;body&gt;</code>
<code>onselect</code>	Seleccionar un texto	<code>&lt;input&gt;</code> , <code>&lt;textarea&gt;</code>
<code>onsubmit</code>	Enviar el formulario	<code>&lt;form&gt;</code>
<code>onunload</code>	Se abandona la página (por ejemplo al cerrar el navegador)	<code>&lt;body&gt;</code>

## EVENTOS

Los eventos más utilizados en las aplicaciones web tradicionales son `onload` para esperar a que se cargue la página por completo, los eventos `onclick`, `onmouseover`, `onmouseout` para controlar el ratón y `onsubmit` para controlar el envío de los formularios.

Algunos eventos de la tabla anterior (`onclick`, `onkeydown`, `onkeypress`, `onreset`, `onsubmit`) permiten evitar la "acción por defecto" de ese evento. Más adelante se muestra en detalle este comportamiento, que puede resultar muy útil en algunas técnicas de programación.

Las acciones típicas que realiza un usuario en una página web pueden dar lugar a una sucesión de eventos. Al pulsar por ejemplo sobre un botón de tipo `<input type="submit">` se desencadenan los eventos `onmousedown`, `onclick`, `onmouseup` y `onsubmit` de forma consecutiva.