

**JS: Conceptos Básicos**

**Sintaxis**

**Variables**

**Control de Flujo**

Antonio Espín Herranz

# ¿Qué es JavaScript?

- Lenguaje de script multiplataforma
  - **Interpretado**
    - Ejecución de *scripts* sin compilación previa
  - **Tipado dinámico débil**
    - Los tipos son asociados con valores, no con variables
  - **Orientado a eventos**
    - E.g. ejecución de un *script* JS (JavaScript) cuando un usuario hace *click* en un botón HTML
  - **Basado en prototipos**
    - Creación de nuevas clases clonando clases base (**prototipos**) y extendiendo su funcionalidad
- Diseñado para **añadir interactividad a páginas web**
- Creado por Brendan Eich (Netscape Communications)
  - Apareció por primera vez en Netscape Navigator 2.0 (1996)
- La versión estándar, ECMAScript, es mantenida por ECMA International

# ¿Qué puede hacer JavaScript?

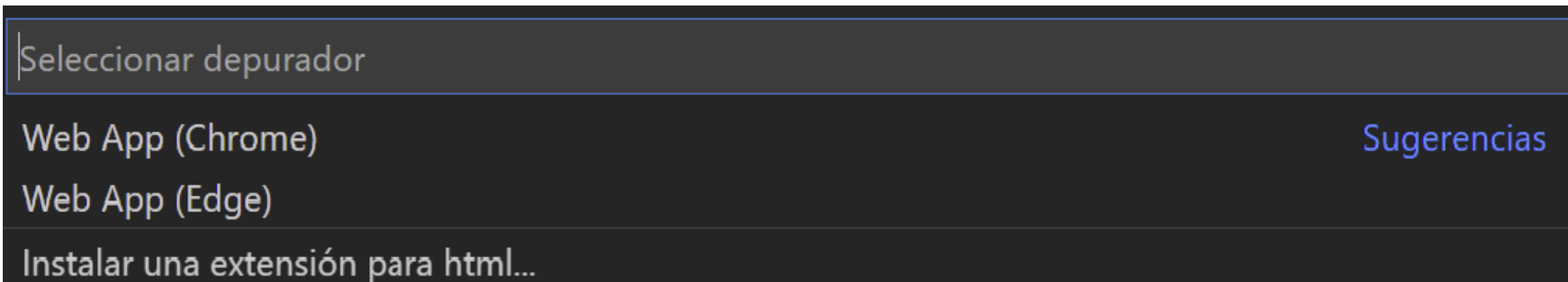
- ❏ JS proporciona a los diseñadores de páginas web una herramienta de programación
  - Diseñadores HTML, pueden insertar fácilmente *snippets* JavaScript
- ❏ **Insertar texto dinámico en una página web**
  - `document.write("<h1>" + name + "</h1>")`
- ❏ Reaccionar a eventos
  - E.g. ejecutar un *script* al terminar de cargar una página HTML o cuando el usuario pulsa un botón
- ❏ **Leer y cambiar el contenido de un elemento HTML → API DOM.**
- ❏ **Validación** de datos
  - Validar un formulario HTML antes de realizar el *submit* al servidor
- ❏ Detectar el navegador del usuario
  - Útil para cargar código específico de un navegador determinado
- ❏ Interfaces ricos (e.g. *drag-n-drop*)

# Introducción

- JavaScript se puede **ejecutar de dos formas**:
  - Utilizando **Node.js** es un framework de javascript que entre otras cosas permite crear un entorno de ejecución para JS en consola. Y se ejecutan los scripts desde la propia consola sin tener que hacer una página.
    - <https://nodejs.org/es/>
  - La otra forma: dentro de una página Web. Adjuntado un fichero de javascript en el head de la página o dentro de una etiqueta `<script>` añadir funciones y declaraciones.

# Ejecución HTML con VS Code

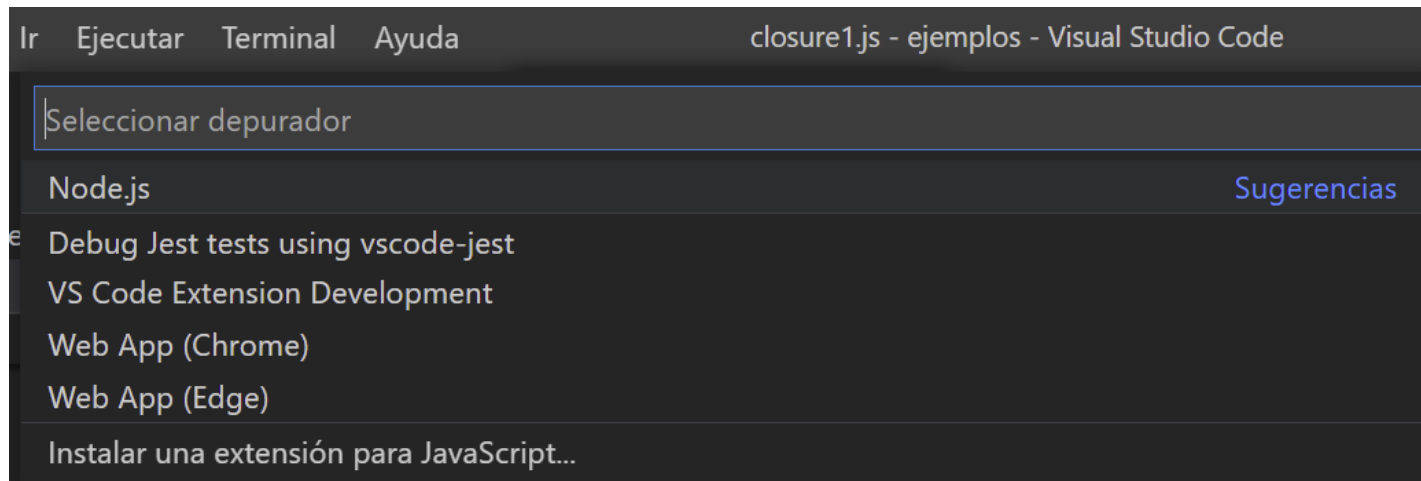
- Desde un **navegador**, con el código JS dentro de las etiquetas script. Se ejecutará normalmente en algún evento.
  - Por ejemplo, al cargar la página o pulsar un botón.
- Dentro de **code** también se puede seleccionar Web App.



Dentro del navegador seleccionar herramientas de desarrollo y buscar consola  
Para ver el resultado si hemos utilizado: `console.log('mensaje')`

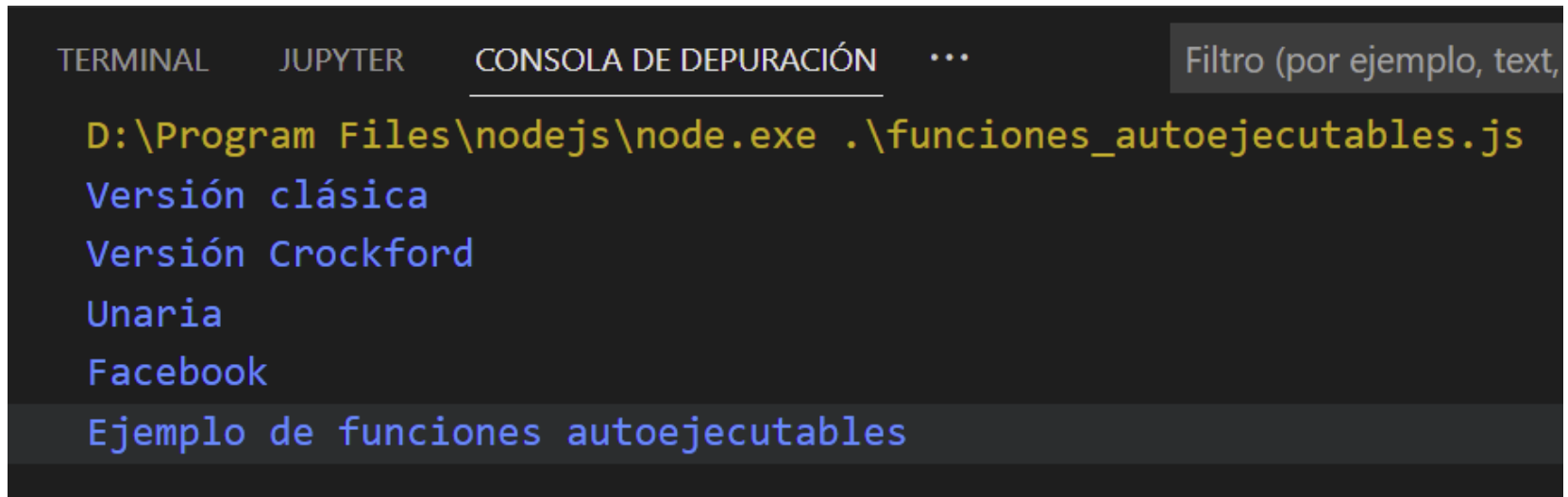
# Ejecución JS con VS. Code

- Para ejecutar el código de JS, tenemos 2 formas:
  - Con el **fichero JS**:
    - Desde un fichero JS y en la **consola** de Windows:  
**node fichero.js**
    - Desde **Visual Studio Code**, seleccionamos **Node.js**



# Ejecución JS con VS. Code

- Desde **Visual Studio Code** podemos ver la salida del script en:

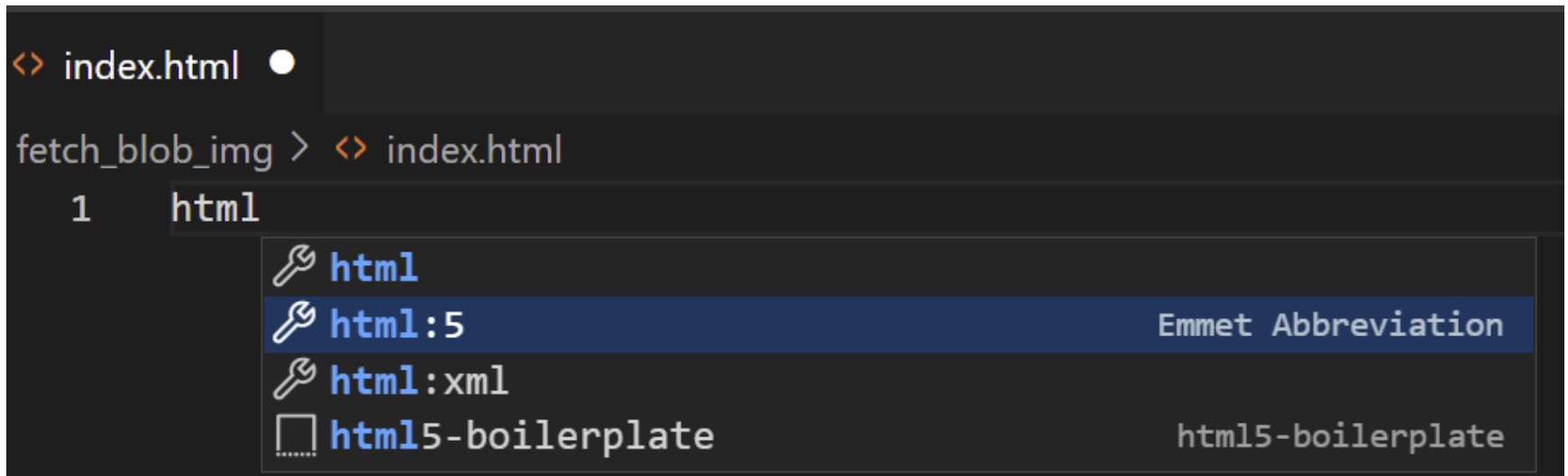


The screenshot shows the Visual Studio Code interface with the 'CONSOLA DE DEPURACIÓN' (Debug Console) tab selected. The terminal displays the command 'D:\Program Files\nodejs\node.exe .\funciones\_autoejecutables.js' and its output, which lists several JavaScript versions and an example of self-executing functions. The output is as follows:

```
D:\Program Files\nodejs\node.exe .\funciones_autoejecutables.js
Versión clásica
Versión Crockford
Unaria
Facebook
Ejemplo de funciones autoejecutables
```

# Template html:5 en VS.Code

- Para generar ficheros html en code: **Crear fichero index.html** y **teclear: html** → seleccionar plantilla **html:5**
  - También, podemos pulsar **!Tab** y tiene el mismo efecto que → html:5



The screenshot shows the VS Code editor with a file named 'index.html' open. The cursor is at the first line, which contains the text 'html'. An Emmet menu is displayed, showing four options: 'html', 'html:5', 'html:xml', and 'html5-boilerplate'. The 'html:5' option is highlighted, and the text 'Emmet Abbreviation' is visible to its right. The 'html5-boilerplate' option has a small icon to its left and the text 'html5-boilerplate' to its right.

```
<> index.html •  
fetch_blob_img > <> index.html  
1 html  
  html  
  html:5 Emmet Abbreviation  
  html:xml  
  html5-boilerplate html5-boilerplate
```



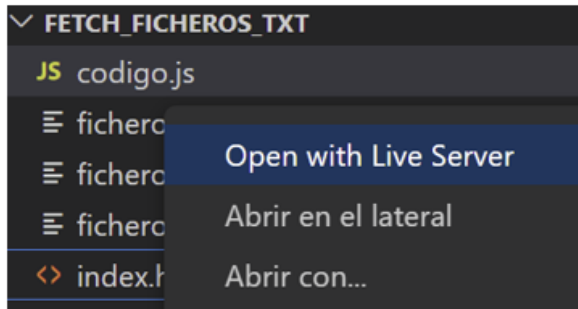
# Ejecutar en Live Server

- VS Code dispone del plugin **Live Server** para visualizar los ficheros html en un servidor local en el puerto 5500.
  - <http://127.0.0.1:5500>
- Se instala en las extensiones.
  - Se utiliza para la parte de peticiones fetch, ejecutar dentro del servidor.
  - Se puede arrancar haciendo click en la barra de status de VS.Code.



# Ejecutar en Live Server

- Para ejecutar también, sobre el fichero: html.



También con el botón derecho sobre el HTML.  
**Open with Live Server**

# ¿Dónde colocar el código JavaScript en una página HTML?

- Entre *tags* `<script>` con el atributo `type`:

```
<html>
<head>
    <script type="text/javascript">
        ...
    </script>
</head>
<body>
    <script type="text/javascript">
        ...
    </script>
</body>
</html>
```

# En *browsers* antiguos...

- ...que no soportan JavaScript, para prevenir mostrar el código JS como contenido, se encierra el código entre comentarios HTML (precediendo el fin de comentario con un doble *slash*): `<!-- ...  
//-->`

```
<html>
<head>
  <script type="text/javascript">
    <!--
      ...
    //-->
  </script>
</head>
<body>
  ...
</body>
</html>
```

# ¿Dónde colocar los scripts? (1)

- Se pueden insertar tanto en la sección `<head>` como en la sección `<body>` de la página HTML (XHTML)

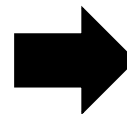
```
<html>
<head>
  <script type="text/javascript">
    function message() {
      alert("Hola mundo!");
    }
  </script>
</head>
<body onload="message()"> </body>
</html>
```

```
<html>
<head>
</head>
<body>
  <script type="text/javascript">
    alert("Hola mundo!");
  </script>
</body>
</html>
```

# ¿Dónde colocar los scripts?

- Generalmente es mejor colocarlos en la sección `<head>`, así nos aseguramos que todo el código llamado desde el `<body>` está cargado (e.g. funciones JS)

```
<html>
<head>
</head>
<body>
  <script type="text/javascript">
    displayMessage();
  </script>
  ...
  <script type="text/javascript">
    function displayMessage(){
      alert("Hola mundo!");
    }
  </script>
</body>
</html>
```



## ERROR:

"displayMessage is not defined"

# Referenciando ficheros JavaScript

- Se puede referenciar un fichero de scripts externo proporcionando su URI en el atributo `src`

Local y Remoto:

```
<html>
<head>
  <script type="text/javascript"
    src="http://unsitioweb/unFicheroJS.js">
  </script>
  <script type="text/javascript"
    src="miDirectorio/miFicheroJS.js">
  </script>
</head>
...
```

# JavaScript y Accesibilidad

- No es necesario eliminar todo el JavaScript para tener un sitio web accesible. **Se puede tener** JavaScript accesible **siguiendo las siguientes** recomendaciones:
  - ▣ Una página tiene que tener el mismo contenido con y sin JavaScript
    - E.g.: no bloquear al usuario que no tenga activado JS
  - ▣ Contenido (i.e. elementos HTML) que sólo tenga sentido con JS activado, tiene que ser creado por el propio JS
    - E.g. :un *link* que sin JS no hace nada `javascript:xxx`)
  - ▣ La funcionalidad del sitio tiene que ser independiente del uso de JS
    - E.g. : si hay un JS que proporciona *drag-n-drop*, esa funcionalidad tiene que proporcionarse también sin JS, por ejemplo, vía *click*)



# Sintaxis...

- Similar a la de otros lenguajes como Java y C
- *Case sensitive*: distingue mayúsculas y minúsculas
- Comentarios:
  - De bloque `/* . . . */`
  - De línea `//`
- Pueden delimitarse bloques de código (e.g. `if`, `for`, ...) con las llaves `{ . . . }`
- Fin de sentencia: nueva línea o un punto y coma ;
- Ignora espacios en blanco *extra*

```
color="azul";  
//equivale a  
color = "azul";
```

- Continuación de cadena de texto mediante una barra invertida: \

```
document.write("Hola \  
    mundo!");
```

# Variables

- ❏ No se define el tipo
- ❏ Una misma variable puede almacenar distintos tipos durante la ejecución de un *script*
- ❏ Declaración explícita (no se tiene en cuenta el ámbito):
  - `var total = 0;`
- ❏ Declaración explícita con `let` (teniendo en cuenta el ámbito):
  - `let total = 0; //mismo efecto que var total = 0;`
- ❏ El alcance de una variable es local a la función en la que se ha declarado (sólo esa función puede accederla)
- ❏ Si una variable es declarada fuera de una función, su ámbito es global a todas las funciones de la página (todas pueden accederla)
- ❏ El tiempo de vida de una variable global comienza cuando se declara y finaliza cuando se cierra la página
  - El de una variable local a una función finaliza cuando se sale de la función

# Tipado

- Javascript pertenece al grupo de lenguajes imperativos pero, sin embargo, se caracteriza por estructuras de tipado débil donde **la declaración de variables no exige la asociación con un tipo de datos de forma implícita y unívoca.**
- **Un tipado blando (o no tipado) significa que las variables son declaradas sin un tipo:** los valores pueden modificarse, compararse y operar entre ellos sin necesidad de realizar una conversión previa.
- **var** x = 15; *// Number (int) declaration*
- **var** y = 15.6 *// Number (float) declaration*
- **var** z = 'Hello World' *// String declaration*
- **var** arr = []; *// Array literal declaration*
- **var** obj = {}; *// Object literal declaration*
- **Javascript asigna el tipo de forma interna:**
- console.log( **typeof**( x ) ); *// number*
- console.log( **typeof**( z ) ); *// string*
- console.log( **typeof**( arr ) ); *// object*
- console.log( **typeof**( obj ) ); *// object*

# Conversiones de Tipos - implícitas

- El intérprete de JS realiza conversiones entre tipos de forma implícita que pueden afectar el resultado de las operaciones.
  - `console.log( 7 + 7 + 7 ); // 21`
  - `console.log( 7 + 7 + "7" ); // "147"`
  - `console.log( "7" + 7 + 7 ); // "777"`
- Las operaciones aritméticas de *adición* (suma) se realizan siguiendo un **orden estricto de izquierda a derecha**. Si un operando corresponde a un tipo de datos string (cadena), el operador '+' lo concatena en lugar de adicionarlo (sumarlo) y todos los operadores posteriores son convertidos en más cadenas que se concatenan sucesivamente.
- Si en una operación de suma uno de los operandos es una cadena, el tipo de datos final será otra cadena con el valor acumulado, y los siguientes, concatenados.

# Conversiones de Tipos - implícitas

- Para el resto de operaciones básicas ( resta, multiplicación y división ), **Javascript siempre trata de convertir en primer lugar las cadenas en números**, por lo que los resultados resultan más intuitivos:
  - `console.log( 10 - 2 - 4 ); // 4`
  - `console.log( 10 - 2 - "4" ); // 4`
  - `console.log( "10" - 2 - 4 ); // 4`
- Cuando operamos en bases diferentes al sistema decimal:
  - `console.log( 1 + 012 ); // 11`

# Conversiones de Tipos - explícitas

- Se pueden forzar los tipos utilizando casting como en otros lenguajes:
- **var** foo = '5'; // En todas el tipo final: **number**
- console.log( **typeof** parseInt( foo ) );
- console.log( **typeof** parseFloat( foo ) );
- console.log( **typeof** ( foo - 0 ) );
- console.log( **typeof** ( foo \* 1 ) );
- console.log( **typeof** ( foo / 1 ) );
- console.log( **typeof** ( +foo ) );

# Conversiones de Tipos - explícitas

- **var** foo = 1;
- console.log( **typeof** ( foo.toString() ) ); // *string*
- Para pasar un número o una cadena a un objeto Booleano, podemos utilizar el recurso de la doble negación '!!':
  - **var** foo = 'Hello World';
  - console.log( !foo ); // *false*
  - console.log( !!foo ); // *true*

# Conversiones de Tipos - explícitas

- También podemos utilizar las funciones:
  - Boolean(value) – casts a Boolean
  - Number(value) – casts a number (integer o real)
  - String(value) – casts a String.
- **Ejemplos:**
  - var value = "true";
  - var value2 = "455";
  - var value3 = 600;
  - console.log( typeof( Boolean(value)) );
  - console.log( typeof( Number(value2)) );
  - console.log( typeof( String(value3)) );



# Ejemplos

- `var b1 = Boolean(""); //false – empty string`
- `var b2 = Boolean("hi"); //true – non-empty string`
- `var b3 = Boolean(100); //true – non-zero number`
- `var b4 = Boolean(null); //false - null`
- `var b5 = Boolean(0); //false - zero`
- `var b6 = Boolean(new Object()); //true – object`
  
- `Number(false) 0`
- `Number(true) 1`
- `Number(undefined) NaN`
- `Number(null) 0`
- `Number("5.5") 5.5`
- `Number("56") 56`
- `Number("5.6.7") NaN`
- `Number(new Object()) NaN`
- `Number(100) 100`

```
var s1 = String(null); //"null"  
var oNull = null;  
var s2 = oNull.toString(); //Error.
```

# Variables / constantes

- Comparativa: var / let
- Con **var**:
  - Se pueden definir variables dentro de ámbitos (por ejemplo, dentro de un bucle for), pero no guarda el ámbito, esa variable va a existir fuera del bucle aunque se haya definido dentro del bucle.
- Con **let**:
  - Si se tiene en cuenta el ámbito donde se define la variable y tiene el mismo comportamiento que en otros lenguajes.
- *Hacer la prueba a definir una variable con var y let dentro de un ámbito y comprobarlo.*

# var vs let

```
var x = 1;
```

```
let y = 1;
```

```
if (true) {
```

```
  var x = 2;
```

```
  let y = 2;
```

```
}
```

```
console.log(x);
```

```
// expected output: 2
```

```
console.log(y);
```

```
// expected output: 1
```

# Constantes

- Se utiliza la palabra reservada **const**
- Es obligatorio definir el valor al declararla.
- Es de solo lectura, no se puede modificar
- `const MAX_SIZE = 4;`

# Operadores

■ **De asignación:** símbolo =

■ **Aritméticos:** suma(+), resta(-), multiplicación(\*), división(/) y resto de división o módulo (%)

- `resultado = operando1 + operando2;`
- `resultado += valor; //asignación y aritmético`
- `variable++; //utiliza el valor de variable y suma 1`
- `++variable; //suma 1 al valor de variable y lo utiliza`

■ **Concatenación de cadenas:** +

■ **Relacionales:**

- Devuelven `true` ó `false`
- `>` Mayor que
- `>=` Mayor o igual
- `<` Menor que
- `<=` Menor o igual

# Operadores

## Aritméticos de notación corta:

-  Es la combinación del operador matemático y el signo igual.

 +=    -=    \*=    /=    %=

## Aritméticos a nivel de bits:

 And &    Xor ^    Or |

# Operadores

## Relacionales (cont)

- `==` Igual que (mismo valor sin ser obligatoriamente mismo tipo)
- `===` Estrictamente igual que (mismo valor y mismo tipo de variable)
- `!=` Distinto que

## Lógicos:

- Evalúan expresiones
- `&&` AND
- `||` OR
- `^` XOR: OR exclusivo (o uno, o lo otro, pero no los dos a la vez)
- `!` NOT: Negación

## Especiales:

- Condicional (`?`):
  - `condicion ? then : else`
  - E.g. `var estado = (edad >= 18) ? "adulto" : "menor";`

# Precedencia de Operadores

() [] . Paréntesis, corchetes y el operador punto que sirve para los objetos

! - ++ -- negación, negativo e incrementos

\* / % Multiplicación división y módulo

+ - Suma y resta

<< >> >>> Cambios a nivel de bit

< <= > >= Operadores condicionales

== != Operadores condicionales de igualdad y desigualdad

& ^ | Lógicos a nivel de bit

&& || Lógicos booleanos

= += -= \*= /= %= <<= >>= >>>= &= ^= != Asignación



# Operadores

## ■ Especiales (cont.)

- Operador **in**
  - Devuelve `true` si la propiedad está especificada en el objeto indicado:
  - propiedad **in** objeto;
- Operador **instanceof**
  - Devuelve `true` si el objeto indicado es una instancia de la clase que se especifica:
  - objeto **instanceof** clase;
- Operador **new**
  - Se utiliza para crear nuevas instancias de objetos
  - `variable = new objeto(param1, param2, ...);`
- Operador **this**
  - Se utiliza más que nada dentro de la definición de un objeto para referirse a una instancia de un objeto.
- Operador **typeof**
  - Devuelve un `string` con el tipo de variable que resulte ser: `undefined`, `boolean`, `number`, `string`, `object`
  - **typeof** variable;

# Sentencias condicionales

- `if`
- `if ... else`
- `if...else if....else`
- `switch`

```
if (hora>=6 && hora<12) {  
    saludo = "buenos días";  
} else if (hora>=12 && hora<21) {  
    saludo = "buenos tardes";  
} else {  
    saludo = "buenas noches";  
}
```

```
theDay=d.getDay();  
switch (theDay) {  
    case 5:  
        document.write("Viernes");  
        break;  
    case 6:  
        document.write("Sábado");  
        break;  
    case 0:  
        document.write("Domingo");  
        break;  
    default:  
        document.write("Esperando el finde");  
}
```

# Sentencias condicionales

- ▣ Operador ternario `? :`
- ▣ Su sintaxis es la siguiente:
- ▣ `expresión_condicional ? expresion1 : expresion2;`
- ▣ `var valor_x_limitado_a_1000 = (x < 1000) ? x : 1000;`
- ▣ Este operador también se puede utilizar en el valor que devuelve una función.

# Bucles (1)

## for

```
var i=0;
for (i=0;i<=10;i++) {
    document.write("Número " + i + "<br/>");
}

for(;;);
```

## while

```
var i=0;
while (i<=10) {
    document.write("Número " + i + "<br/>");
    i++;
}
```

## do ... while

```
var i=0;
do {
    document.write("Número " + i + "<br/>");
    i++;
} while (i<0);
```

*//sólo imprimirá Número 0 (1ª iteración)*

# Ejemplo bucle for

- **Generar:**
  - `<H1>Encabezado de nivel 1</H1>`
  - `<H2>Encabezado de nivel 2</H2>`
  - `<H3>Encabezado de nivel 3</H3>`
  - `<H4>Encabezado de nivel 4</H4>`
  - `<H5>Encabezado de nivel 5</H5>`
  - `<H6>Encabezado de nivel 6</H6>`
- 
- ```
for (i=1;i<=6;i++) {
```
  - ```
    document.write("<H" + i + ">Encabezado de nivel " + i + "</H" + i + ">")
```
  - ```
}
```

# Bucles (2)

## break

- Rompe el bucle y continúa la ejecución fuera del mismo

```
var i=0;
for (i=0;i<=10;i++) {
    if (i ==3) {
        break;
    }
    document.write("Número " + i + "<br/>");
}

//sólo imprime los números 0, 1 y 2
```

## continue

- Rompe la iteración actual y continúa en la siguiente

```
var i=0;
for (i=0;i<=10;i++) {
    if (i ==3) {
        continue;
    }
    document.write("Número " + i + "<br/>");
}

//imprime los números 0, 1, 2, 4, 5, ..., 10
```

# Bucles (y 3)

## **for ... In / for ... of**

- Itera sobre los elementos de un array o las propiedades de un objeto

```
var x;  
var marcasCoches = new Array();  
marcasCoches[0] = "Volkswagen";  
marcasCoches[1] = "Mitsubishi";  
marcasCoches[2] = "Honda";
```

```
for (x in marcasCoches) {  
    console.log(marcasCoches[x]);  
}
```

```
for (x of marcasCoches) {  
    console.log(x);  
}
```

# console

- El objeto console se utiliza en JS entre otras cosas para imprimir mensajes por la consola.
  - Estos mensajes se pueden ver por una consola de Windows / Linux si estamos ejecutando el script con Node.js
  - O por la consola del navegador, habrá que entrar en algún menú (opciones del desarrollador) para verla.



# Métodos de console (standard)

|                                       |                                                                                                                                                     |
|---------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>console.assert()</code>         | Imprime un error en la consola si la afirmación es falsa                                                                                            |
| <code>console.clear()</code>          | Limpia la consola                                                                                                                                   |
| <code>console.count()</code>          | Registra el número de veces que se ha llamado esta llamada particular a <code>count()</code> . Esta función toma una etiqueta de argumento opcional |
| <code>console.error()</code>          | Imprime un error en la consola                                                                                                                      |
| <code>console.group()</code>          | Crea un grupo de mensajes en la consola                                                                                                             |
| <code>console.groupCollapsed()</code> | Crea un grupo de mensajes contraído en la consola. El usuario deberá desplegarlo para ver su contenido                                              |
| <code>console.groupEnd()</code>       | Cierra un grupo de mensajes en la consola                                                                                                           |
| <code>console.info()</code>           | Muestra información sobre la consola                                                                                                                |
| <code>console.log()</code>            | Imprime variables y/o cadenas en la consola del navegador                                                                                           |
| <code>console.table()</code>          | Imprime información con formato tabla. Por ejemplo, un array                                                                                        |
| <code>console.time()</code>           | Inicia un cronometro                                                                                                                                |
| <code>console.timeEnd()</code>        | Para el cronómetro                                                                                                                                  |
| <code>console.trace()</code>          | Imprime un seguimiento de la pila de consola                                                                                                        |
| <code>console.warn()</code>           | Imprime un mensaje de warning en la consola                                                                                                         |

# Métodos de console (Non-standard)

|                                   |                                                                                                                                                                                        |
|-----------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>console.dir()</code>        | Muestra una lista interactiva del listado de propiedades de un objeto JavaScript                                                                                                       |
| <code>console.dirxml()</code>     | Muestra un árbol interactivo de los elementos descendientes del elemento XML / HTML especificado                                                                                       |
| <code>console.profile()</code>    | Comienza a grabar un perfil de rendimiento                                                                                                                                             |
| <code>console.profileEnd()</code> | Para la grabación del perfil de rendimiento                                                                                                                                            |
| <code>console.timeStamp()</code>  | Agregue un solo marcador a la herramienta Timeline o Waterfall del navegador. Esto le permite correlacionar un punto en su código con los otros eventos grabados en la línea de tiempo |

# Ejemplos

- `console.log('Consola de Registro')`
- `console.info('Consola de Informacion')`
- `console.debug('Console de Depuracion')`
- `console.warn('Consola de Aviso')`
- `console.error('Consola de Error')`

# Enlaces

- Con respecto a la consola:
  - <https://www.freecodecamp.org/espanol/news/ejemplo-de-console-log-en-javascript-como-imprimir-en-la-consola-en-js/>