

DESARROLLO WEB EN ENTORNO CLIENTE

CAPÍTULO 2:

Introducción al lenguaje JavaScript

Objetivos del capítulo

- Conocer las principales características del lenguaje JavaScript.
- Dominar la sintaxis básica del lenguaje.
- Comprender y utilizar los distintos tipos de variables y operadores presentes en el lenguaje JavaScript.
- Conocer las diferentes sentencias condicionales de JavaScript y saber realizar operaciones complejas con ellas.

ÍNDICE

1. Historia de JavaScript
2. Características de JavaScript
3. “Hola mundo” con JavaScript
4. El lenguaje JavaScript: sintaxis
5. Entrada y salida de datos
6. Variables
7. Tipos de datos
8. Operadores
9. Estructuras de control

ÍNDICE

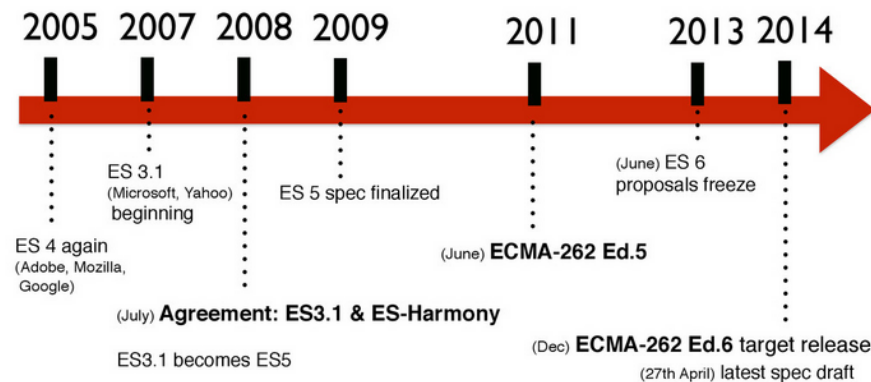
1. **Historia de JavaScript**
2. Características de JavaScript
3. “Hola mundo” con JavaScript
4. El lenguaje JavaScript: sintaxis
5. Entrada y salida de datos
6. Variables
7. Tipos de datos
8. Operadores
9. Estructuras de control

Historia de JavaScript

- Desarrollado por **Brendan Eich** de **Netscape** con el nombre de *Mocha* en **1995**. Para solucionar el problema de navegar por internet con una velocidad tan lenta. Propuso crear un lenguaje que se ejecutara en el cliente (navegador web).
- Renombrado posteriormente a *LiveScript*, para finalmente quedar como **JavaScript**
- El cambio de nombre coincidió aproximadamente con el momento en que Netscape agregó soporte para la tecnología Java en su navegador web Netscape Navigator en la versión 2.002 en diciembre de 1995.
- Produjo **confusión**, dando la impresión de que el lenguaje es una prolongación de Java, y se ha caracterizado por muchos como una estrategia de Netscape para obtener prestigio e innovar en lo que eran los nuevos lenguajes de programación web.

Historia de JavaScript

- **Microsoft** lanzó su propia versión JavaScript «**JScript**» Los dialectos pueden parecer tan similares que los términos «JavaScript» y «JScript» a menudo se utilizan indistintamente, pero la especificación de JScript es incompatible con la de ECMA en muchos aspectos.
- En respuesta a estas incompatibilidades, la **ECMA (European Computer Manufacturers Association)** emprendió un esfuerzo de estandarización que desembocó en la publicación del **estándar ECMAScript**.
- A día de hoy, tanto JavaScript como JScript son conformes a dicho estándar, es el término JavaScript el que se impuso y se utiliza para referirse al propio lenguaje y al estándar.



Historia de JavaScript

- ECMAScript 6 quedó cerrado en junio 2015. El nombre oficial del lenguaje es ahora **ECMAScript 2015**. Aportaciones:
 - **Mejoras de sintaxis**: parámetros por defecto, let, plantillas...
 - **Módulos para organización de código**
 - **Verdaderas clases** para programación orientada a objetos
 - Promesas, para **programación asíncrona**
 - **Mejoras en programación funcional**: expresiones de flecha, iteradores, generadores...
- **ECMAScript 7**, cuyo nombre oficial es **ECMAScript 2016**. Añade: básicamente el **operador de exponenciación** y un método nuevo para las matrices que permite comprobar si existen ciertos elementos dentro de éstas.
- **ES8** Constructores async/await para generadores y promesas
- **ECMAScript 2018**, incluye **operadores rest/spread para variables** (tres puntos: ...identificador), **iteración asíncrona**, **Promise.prototype.finally()**
- La **10.ª edición**, incorporó **Array.flat()**, **Array.flatMap()**, **String.trimStart()**, **String.trimEnd()**, errores opcionales en el **bloque catch**, **Object.fromEntries()**

Historia de JavaScript

Ver	Official Name	Description
1	ECMAScript 1 (1997)	First Edition.
2	ECMAScript 2 (1998)	Editorial changes only.
3	ECMAScript 3 (1999)	Added Regular Expressions. Added try/catch.
4	ECMAScript 4	Never released.
5	ECMAScript 5 (2009)	Added "strict mode". Added JSON support. Added String.trim(). Added Array.isArray(). Added Array Iteration Methods.
5.1	ECMAScript 5.1 (2011)	Editorial changes.
6	ECMAScript 2015	Added let and const. Added default parameter values. Added Array.find(). Added Array.findIndex().
7	ECMAScript 2016	Added exponential operator (**). Added Array.prototype.includes.
8	ECMAScript 2017	Added string padding. Added new Object properties. Added Async functions. Added Shared Memory.
9	ECMAScript 2018	Added rest / spread properties. Added Asynchronous iteration. Added Promise.finally(). Additions to RegExp.

ÍNDICE

1. Historia de JavaScript
2. **Características de JavaScript**
3. “Hola mundo” con JavaScript
4. El lenguaje JavaScript: sintaxis
5. Entrada y salida de datos
6. Tipos de datos
7. Variables
8. Operadores
9. Estructuras de control

Características de JavaScript

- ¿Qué es JavaScript?
- Lenguaje de programación **interpretado** utilizado fundamentalmente para dotar de **comportamiento dinámico** a las páginas web.
- Cualquier navegador web actual incorpora un intérprete para código JavaScript.

Características de JavaScript

- Su **sintaxis** se **asemeja a la de C++ y Java**.
- Está **basado en el concepto de objeto, pero no es un lenguaje orientado a objetos**.
- Sus objetos utilizan **herencia basada en prototipos** -- los objetos no **son creados mediante** la instanciación de clases sino mediante **la clonación de otros objetos o mediante la escritura de código por parte del programador**. De esta forma los objetos ya existentes pueden servir de prototipos para los que el programador necesite crear.
- Es un lenguaje **débilmente tipado**.
- **Por defecto, todas sus variables son globales**.

ÍNDICE

1. Historia de JavaScript
2. Características de JavaScript
3. **“Hola mundo” con JavaScript**
4. El lenguaje JavaScript: sintaxis
5. Entrada y salida de datos
6. Variables
7. Tipos de datos
8. Operadores
9. Estructuras de control

“Hola mundo” con JavaScript

- Es necesario sólo un navegador web y un editor de texto.
- Hay dos formas de embeber el código JavaScript en una página HTML:
 1. Incluirlo directamente en la página HTML mediante la etiqueta `<script>` en la etiqueta `<head>` o `<body>`.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Hola Mundo</title>
  </head>
  <body>
    <script>
      alert('Hola mundo en JavaScript');
    </script>
  </body>
</html>
```

DOCTYPE no es una etiqueta sino una declaración para identificar la version de HTML o XLM

meta charset="UTF-8" especifica la codificación de caracteres del documento. UTF-8 es un conjunto de caracteres universal que incluye casi todos los caracteres de casi cualquier idioma humano

aparece una ventana con el mensaje

¿Dónde aparece el nombre del TITULO que se ha puesto en la cabecera?

“Hola mundo” con JavaScript

2. Utilizar el atributo `src` de la etiqueta `<script>` para especificar el fichero que contiene el código JavaScript.

```
<!DOCTYPE html>
```

```
<html>
```

```
  <head>
```

```
    <meta charset="UTF-8">
```

```
    <title>Hola Mundo</title>
```

```
    <script type="text/javascript" src="HolaMundo.js">
```

```
  </script>
```

```
</head>
```

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

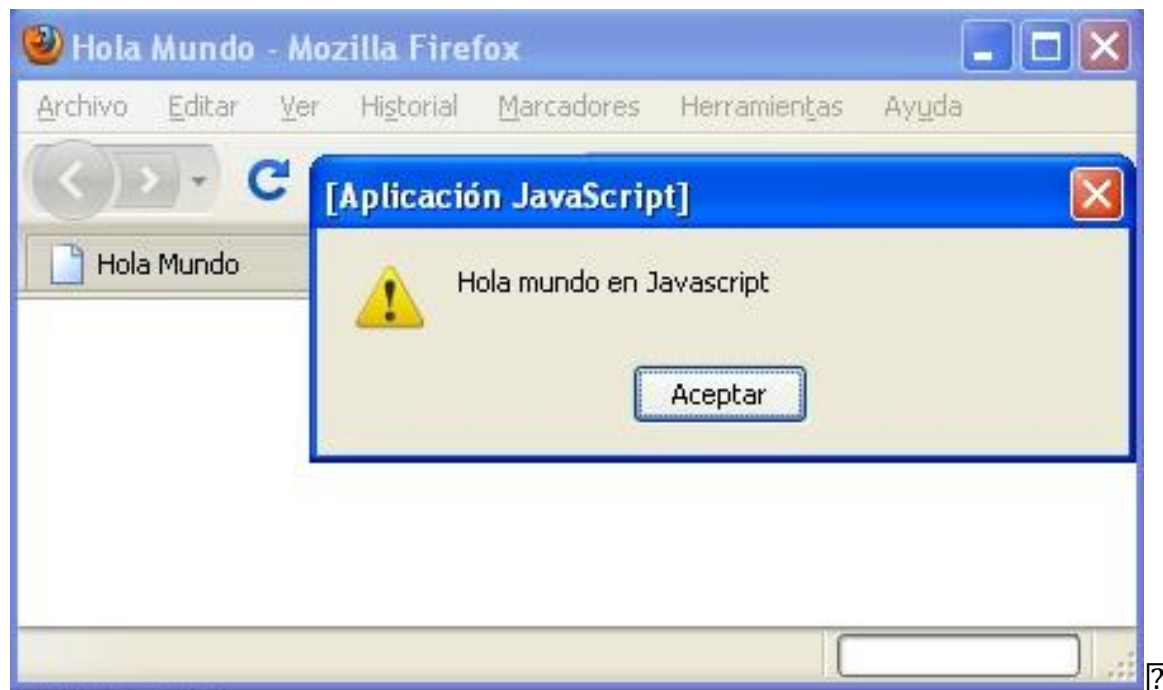
```
</html>
```

INTENTAR HACER ESTE CODIGO Y VER EL
RESULTADO

https://www.w3schools.com/js/js_editor.asp

El fichero `HolaMundo.js` debe contener:
`alert('Hola Mundo en JavaScript')`

“Hola mundo” con JavaScript



Carga diferida de scripts

- Se puede cargar asíncronamente con `async`

```
<script src="el_script.js" async></script>
```

- Se puede cargar un script en diferido con `defer`

```
<script src="el_script.js" defer></script>
```

- Para ‘minimizar’ el tiempo de carga, carga en diferido:

```
<script type="text/javascript">
    function downloadJSAtOnload() {
        var element = document.createElement("script");
        element.src = "el_script.js";
        document.body.appendChild(element);
    }
    if (window.addEventListener)
        window.addEventListener("load", downloadJSAtOnload, false);
    else if (window.attachEvent)
        window.attachEvent("onload", downloadJSAtOnload);
    else window.onload = downloadJSAtOnload;
</script>
```


ÍNDICE

1. Historia de JavaScript
2. Características de JavaScript
3. “Hola mundo” con JavaScript
4. **El lenguaje JavaScript: sintaxis**
5. Entrada y salida de datos
6. Variables
7. Tipos de datos
8. Operadores
9. Estructuras de control

El lenguaje JavaScript: sintaxis

- La sintaxis de JavaScript es muy similar a la de Java o C++.
- **Mayúsculas y minúsculas:**
 - Es *case sensitive* a diferencia de por ejemplo HTML
 - No es lo mismo utilizar alert() que Alert().

El lenguaje JavaScript: sintaxis

- **Comentarios en el código:**

- Los comentarios no se interpretan por el navegador.
- Existen **dos formas** de insertar comentarios:
 - Doble barra (//) – Se comenta **una sola línea** de código.
 - Barra y asterisco (/ * al inicio y */ al final) – Se comentan **varias líneas** de código.

```
<script type="text/javascript">  
    // Este modo permite comentar  
    una sola línea  
    /* Este modo permite realizar  
    comentarios de  
    varias líneas */  
</script>
```

El lenguaje JavaScript: sintaxis

- **Tabulación y saltos de línea:**

- JavaScript ignora los espacios, las tabulaciones y los saltos de línea con algunas excepciones.
- Emplear la tabulación y los saltos de línea mejora la presentación y la legibilidad del código.

```
<script
type="text/javascript">var
i,j=0;
for (i=0;i<5;i++){
alert("Variable i: "+i);
for (j=0;j<5;j++){ if
(i%2==0){
document.write
(i + "-" + j +
"<br>");}}</script>
```

```
<script type="text/javascript">
var i,j=0;
for (i=0;i<5;i++){
    alert("Variable i: "+i;
    for (j=0;j<5;j++){
        if (i%2==0){
            document.write(i + "-" + j + "<br>");
        }
    }
}
</script>
```

El lenguaje JavaScript: sintaxis

- **El punto y coma:**

- Se suele insertar un signo de punto y coma (;) al final de cada instrucción de JavaScript, aunque no es obligatorio.
- Su utilidad es separar y diferenciar cada instrucción.
- Se puede omitir si cada instrucción se encuentra en una línea independiente (la omisión del punto y coma no es una buena práctica de programación).

```
pi_egipcio = 3.16  
pi_griego = 3.14
```

```
pi_egipcio = 3.16 ; pi_griego = 3.14;
```

El lenguaje JavaScript: sintaxis

- **Palabras reservadas:**

- Algunas palabras no se pueden utilizar para definir nombres de variables, funciones o etiquetas.
- Es aconsejable no utilizar tampoco las palabras reservadas para futuras versiones de JavaScript.
- En [ecma-internacional](http://www.ecma-international.org) es posible consultar todas las palabras reservadas de JavaScript.



Visitar la página web y consultar todas las palabras reservadas actuales y para las futuras versiones del estándar.

ÍNDICE

1. Historia de JavaScript
2. Características de JavaScript
3. “Hola mundo” con JavaScript
4. El lenguaje JavaScript: sintaxis
5. **Entrada y salida de datos**
6. Variables
7. Tipos de datos
8. Operadores
9. Estructuras de control

Salida: Ventana “ALERT”

- Se trata de una ventana estándar que usamos para mostrar información en pantalla. Se puede mostrar texto, variables y texto en conjunto con variables. Si se quiere mostrar texto se pone entre comillas.

SINTAXIS:

`alert(“texto de la ventana”);`

`alert(variable);`

`alert(“texto”+variable);`



Nota: `alert("Hola")` es igual `window.alert("Hola")`

Salida: `write` y `innerHTML`

- **`document.write("texto")`**: Opción para mostrar una salida directamente dentro del documento HTML. Se recomienda su uso solo para pruebas. Introduce , aunque admite expresiones y código HTML:

```
document.write("<p>Esto es un párrafo</p>");
```

- **`document.getElementById("id").innerHTML`**: Opción más recomendable. También admite código HTML:

```
document.getElementById("div2").innerHTML=("<p>Esto es tu nombre:  
"+nombre+"</p>");
```

Salida: write y innerHTML

```
<body>
  <div>
    <h1 id="cabecera">Hola a todos</h1>
    <div id="div1"></div>
    <div id="div2"></div>
  </div>
  <script>
    document.write("<p>Esto es un párrafo introducido por write</p>");
    document.getElementById("div1").innerHTML=("Sección 1");
    var nombre="Damián León";
    alert("Hola " + nombre);
    document.getElementById("div2").innerHTML=("<p>Este es tu nombre: "+nombre+"</p>");
  </script>
</body>
```

Hola a todos

Sección 1

Este es tu nombre: Damián León

Esto es un párrafo introducido por write

Salida: Ventana “Confirm”

- Muestra una ventana de confirmación.
 - Si el usuario acepta → true
 - En caso contrario → false

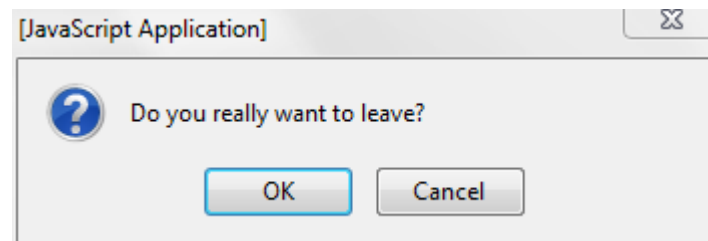
SINTAXIS:

`confirm(“texto de la ventana”);`

```
var c=confirm("Pulsa un boton");  
var texto;
```

```
if (c){  
    texto="aceptar"  
} else {  
    texto="cancelar"  
}
```

```
document.getElementById("div3").innerHTML=("Has pulsado: "+texto);
```

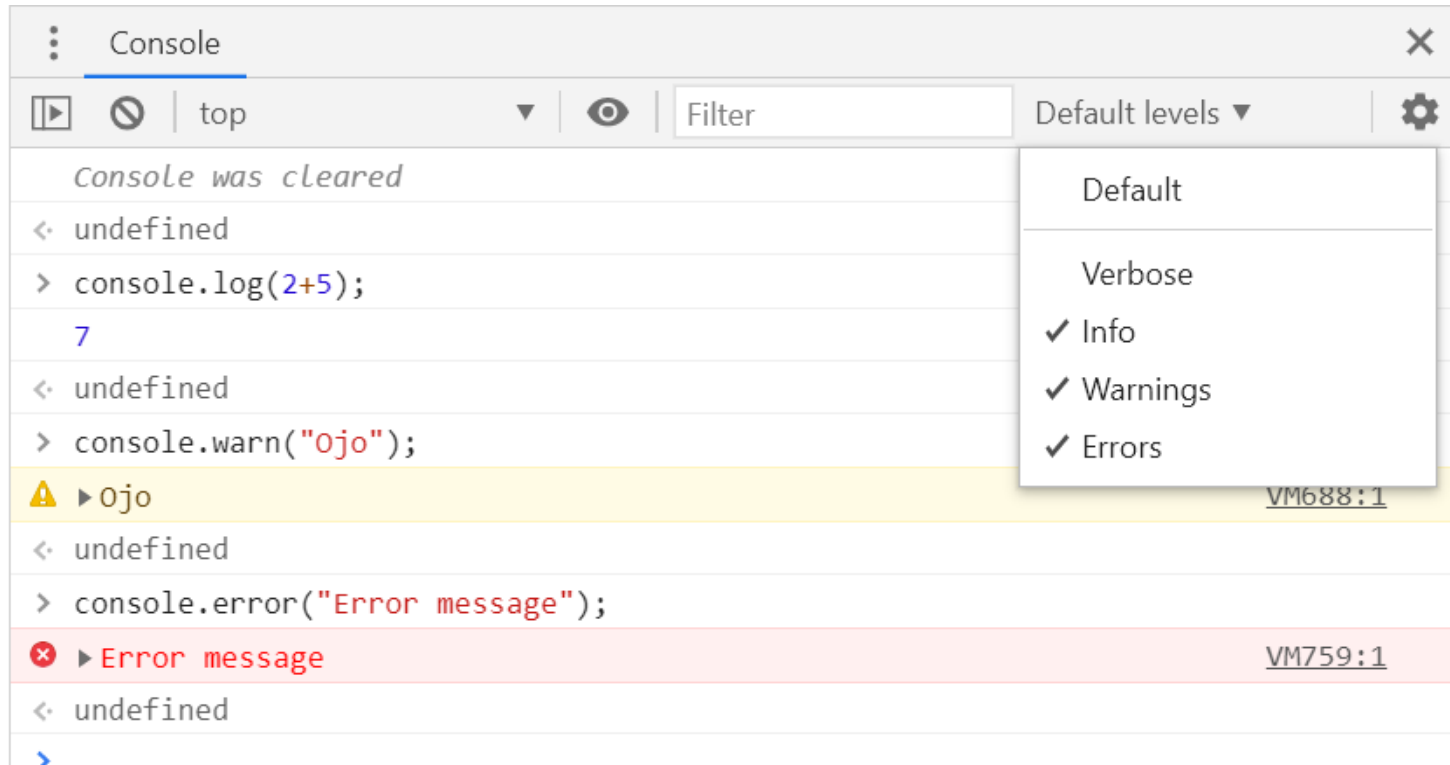


Salida: Consola

- Se pueden utilizar la consola como salida de datos que no se consideran oportunos que aparezca en el HTML para el usuario pero sí que pueden ser útiles como programadores.
- Además la consola permite sentencias en JS
 - `console.log("texto" o expresión);`
 - `console.info("texto");`
 - `console.warning("mensaje warning");`
 - `console.error("mensaje de error");`

Salida: Consola

- Se pueden filtrar el tipo de mensajes desde la consola:



Salida: Consola

- **debugger:** Para la ejecución de la página en un punto específico

The screenshot displays a web browser window on the left and its developer console on the right. The browser window shows the text "Hola a todos" and "Sección 1". Below this, there is a text input field containing "Este es tu nombre: Damián León" and a paragraph of text "Este es un párrafo introducido por write". The developer console on the right is open to the "Sources" tab, showing the file "00.html". The code is paused at line 22, which is `var c=confirm("Pulsa un boton");`. The console also shows the "Debugger paused" message and the "Call Stack" with the function "(anonymous)" at line 22 of "00.html".

Browser Window:

Hola a todos

Sección 1

Este es tu nombre: Damián León

Este es un párrafo introducido por write

Developer Console:

Paused in debugger

00.html

```
13
14
15     <script>
16         debugger;
17         alert("Hola");
18         document.write (<p>Esto es un párrafo intr
19         document.getElementById("div1").innerHTML=(
20
21         var nombre="Damián León";
22         document.getElementById("div2").innerHTML=(
23         var c=confirm("Pulsa un boton");
24         var texto;
25         if (c){
26             texto="aceptar"
27         }else {
28             texto="cancelar"
29         }
30         document.getElementById("div3").innerHTML=(
31
32     </script>
```

Line 22, Column 23

Coverage: n/a

Debugger paused

Call Stack

(anonymous) 00.html:22

Scope Watch

Global

Window

Entrada: Ventana Prompt

- JavaScript permite interactuar al usuario por medio de la introducción de datos
- La introducción de datos se puede realizar por medio de la ventana prompt o utilizando controles como cajas de texto.

- **VENTANA PROMPT:**

- *SINTAXIS:*

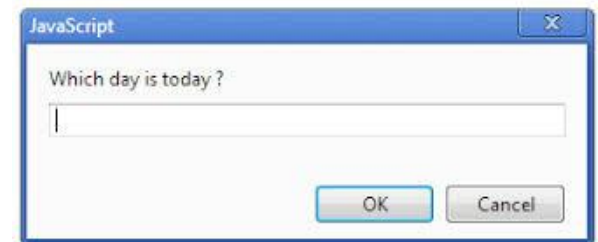
- ```
vari=prompt("Texto de la ventana","valor inicial caja");
```

- Al pulsar el botón aceptar, el contenido de la caja pasa a *vari*. Si se pulsa el botón cancelar, el contenido de la caja se pierde y *vari* queda con valor null.

- **¡Ojo! Siempre devuelve un String**

Para convertirlo en número tendremos que usar

```
x= parseInt (prompt ("Introduce un número: "));
```



# Introducción de Datos “Prompt”

## *EJEMPLO:*

```
<body>
 <div>
 <h1 id="cabecera">Hola a todos</h1>
 <div id="div1"></div>
 <div id="div2"></div>
 <div id="div3"></div>
 </div>
 <script>
 debugger;
 alert("Hola");
 document.write("<p>Esto es un párrafo por write</p>");
 document.getElementById("div1").innerHTML="Sección 1";
 var nombre=prompt("Introduce tu nombre:");
 document.getElementById("div2").innerHTML=("<p>Este es tu nombre:
"+nombre+"</p>");
 var c=confirm("Pulsa un boton");
 var texto;
 if (c){texto="aceptar"}else {texto="cancelar"}
 document.getElementById("div3").innerHTML=("Has pulsado: "+texto);
 </script>
</body>
```



# ÍNDICE

1. Historia de JavaScript
2. Características de JavaScript
3. “Hola mundo” con JavaScript
4. El lenguaje JavaScript: sintaxis
5. Entrada y salida de datos
6. **Variables**
7. Tipos de datos
8. Operadores
9. Estructuras de control

# Variables

- Se pueden definir como zonas de la memoria de un ordenador que se identifican con un nombre y en las cuales se almacenan ciertos datos.
  - El nombre o **identificador** debe cumplir dos condiciones:
    - El *primer carácter* debe ser una **letra** aunque también puede ser un **guión bajo** (**\_**) o un **dólar** (**\$**).
    - El *resto de caracteres* pueden ser letras, números, guiones bajos.
    - Además, JS es sensible a mayúsculas y **no se pueden usar palabras reservadas**.
  - El desarrollo de un script conlleva:
    - Declaración de variables\*.
    - Inicialización de variables.
- \* *No es obligatorio la declaración de variables.*

# Variables

- **Declaración de variables:**
  - Se declaran mediante la palabra clave `var` seguida por el nombre que se quiera dar a la variable.
    - `var mi_variable;`
  - Es posible declarar más de una variable en una sola línea.
    - `var mi_variable1, mi_variable2;`

# Variables

- **Inicialización de variables:**

- Se puede **asignar un valor a una variable** de tres formas:

- Asignación directa de un valor concreto.

```
var mi_variable_1 = 30;
var x=1,y=2,z=3;
```

- Asignación indirecta a través de un cálculo en el que se implican a otras variables o constantes.

```
var mi_variable_2 = mi_variable_1 + 10;
```

- Asignación a través de la solicitud del valor al usuario del programa.

```
var mi_variable_3 = prompt('Introduce un valor:');
```

# Variables

- **Ámbito** (scope)
- **Tiempo de vida de las variables en JavaScript**
  - Comienza desde que es declarada.
  - Las **variables locales** son eliminadas cuando la función es completada. Solo se pueden recuperar dentro de su ámbito, no se pueden recuperar desde consola.
  - Las **variables globales** son eliminadas cuando la página se cierra. Se pueden recuperar desde cualquier punto, por ejemplo desde la consola.

# Variables

- **Ámbito** (scope)
  - Cuando utilizamos **var** estamos haciendo que la variable que estamos declarando sea **local** al ámbito donde se declara.
  - Si no utilizamos la palabra var para declarar una variable, ésta será **global** a toda la página, sea cual sea el ámbito en el que haya sido declarada.
  - En caso de **colisión** entre las variables globales y locales, dentro de una función prevalecen las variables **locales**.

# Variables

- **Ámbito** (scope)

```
function creaMensaje() {
 var mensaje = "Mensaje de prueba";
 alert(mensaje);
}
creaMensaje();
```

```
var mensaje = "Mensaje de prueba";

function muestraMensaje() {
 alert(mensaje);
}
```



```
mensaje = "Mensaje de prueba";

function muestraMensaje() {
 alert(mensaje);
}
```

# Variables

- **Ámbito** (scope)
  - ¿Qué se visualizará por pantalla?

```
function creaMensaje() {
 var mensaje = "Mensaje de prueba";
}
creaMensaje();
alert(mensaje);
```



# Variables

- **Ámbito** (scope)

- Ejemplos:

```
var mensaje = "gana la de fuera";
function muestraMensaje() {
 var mensaje = "gana la de dentro";
 alert(mensaje);
}
alert(mensaje);
muestraMensaje();
alert(mensaje);
```

## Resultado

gana la de fuera  
gana la de dentro  
gana la de fuera

```
var mensaje = "gana la de fuera";
function muestraMensaje() {
 mensaje = "gana la de dentro";
 alert(mensaje);
}
alert(mensaje);
muestraMensaje();
alert(mensaje);
```

## Resultado

gana la de fuera  
gana la de dentro  
gana la de dentro

# Variables



- Dado el siguiente código:

```
<script type="text/javascript">
 var primer_saludo="hola";
 var segundo_saludo=primer_saludo;
 primer_saludo="hello";
 alert(primer_saludo);
 alert(segundo_saludo);
</script>
```

- ¿Cuál será el valor de la primera ventana emergente? ¿Y de la segunda?

# Hoisting: Mueve declaraciones al inicio del su scope

Para evitar errores es mejor declarar las variables al inicio

- En Javascript una variable puede ser declarada después de haber sido usada

```
x = 5; // Assign 5 to x

elem = document.getElementById("demo"); // Find an element
elem.innerHTML = x; // Display x in the element

var x; // Declare x
```

5

- Este lenguaje permite subir las declaraciones de las variables al principio del ámbito pero no las inicializaciones.

```
var x = 5; // Initialize x

elem = document.getElementById("demo"); // Find an element
elem.innerHTML = x + " " + y; // Display x and y

var y = 7; // Initialize y
```

x is 5 and y is undefined

# JavaScript – ES6 novedades Variables

- **Let**

- Permite declarar variables locales limitando su alcance al bloque, declaración o expresión donde se está usando . Dejan de ser globales a la función (var).

```
//ES5
(function() {
 console.log(x); // x no está definida aún.
 if(true) {
 var x = "hola mundo";
 }
 console.log(x);
 // Imprime "hola mundo", porque "var" hace que sea global
 // a la función;
})();

//ES6
(function() {
 if(true) {
 let x = "hola mundo";
 }
 console.log(x);
 //Da error, porque "x" ha sido definida dentro del "if"
})();
```

# JavaScript – ES6 novedades Variables

- **Const**

- No van a modificar su valor en tiempo de ejecución. Podemos crear constantes que sólo se puedan leer y no modificar a lo largo del código

```
const PI;
PI = 3.15;
// ERROR, porque ha de asignarse un valor en la
// declaración
```

```
const PI = 3.15;
PI = 3.14159;
// ERROR de nuevo, porque es sólo-lectura
```

- **const LOCATION = MAYÚSCULAS**
- **const DEFAULT\_LOCATION = MAYÚSCULAS con guiones**

# JavaScript – ES6 novedades Variables

<https://cybmeta.com/var-let-y-const-en-javascript>

- **Resumen**

- **var** declara una variable de scope global o local para la función sin importar el ámbito de bloque. Permite hoisting.
- **let** declara una variable de scope global, local para la función o de bloque. Es reassignable y no permite hoisting.
- **const** declara una variable de scope global, local para la función o de bloque. No es reassignable, pero es mutable. No permite hoisting.

En general, `let` sería todo lo que se necesita dentro de un bloque, función o ámbito global. `const` sería para variables que no van sufrir una reasignación. `var` se puede relegar para cuando necesitemos hacer hoisting, vamos, casi nunca.

# ÍNDICE

1. Historia de JavaScript
2. Características de JavaScript
3. “Hola mundo” con JavaScript
4. El lenguaje JavaScript: sintaxis
5. Entrada y salida de datos
6. Variables
7. **Tipos de datos**
8. Operadores
9. Estructuras de control

# Tipos de datos

- Los tipos de datos especifican qué tipo de valor se guardará en una determinada variable.
- Los cinco tipos de datos primitivos de JavaScript son:
  - undefined
  - null
  - boolean
  - number
  - string



# Tipos de datos

- Operador **typeof** :
  - Devuelve el tipo de dato que almacena una variable.
  - Los posibles valores de retorno del operador son: **undefined**, **boolean**, **number**, **string** para cada uno de los tipos primitivos, **object** para los datos complejos: objetos, tipos de referencia (new String...) y también para los valores de tipo null.
  - Ejemplo:

```
typeof "John" // Returns string
typeof 3.14 // Returns number
typeof false // Returns boolean
typeof [1,2,3,4] // Returns object
typeof {name:'John', age:34} // Returns object
```

# Tipos de datos

- **undefined:**

Pero No inicializadas

- Variables que han sido definidas y todavía no se les ha asignado un valor (Ausencia de valor).

```
var variable1;
```

```
typeof variable1; // devuelve "undefined"
```

- **Nota:** El operador `typeof` no distingue entre las variables declaradas pero no inicializadas y las variables que ni siquiera han sido declaradas

# Tipos de datos

- **null:**

- Similar a undefined, y de hecho en JavaScript se consideran iguales: `undefined == null`, pero no `undefined === null` (Valor de nada, es decir esta definida pero no tiene valor, es nulo).
- El tipo null se suele utilizar para representar objetos que en ese momento no existen
- Ejemplo: `var nombreUsuario = null;`
- Recuerda `typeof(null) === object`

# Tipos de datos

- **number:**
  - En JavaScript existe sólo un tipo de dato numérico.
  - Todos los números se representan a través del formato de punto flotante de 64 bits.
  - Este formato es el llamado **double** en los lenguajes Java o C++.
- Ejemplos:
  - Número entero → 45
  - Número decimal → 3.1415
  - Sistema octal → 034
  - Sistema hexadecimal → 0xA3
- Operaciones: Aritméticas (+, -, \*, /, %, \*\*), de comparación(==, >=...)

# Tipos de datos

- **number:**
  - JavaScript define tres valores especiales :  
Infinity, -Infinity y NaN (Not a Number)
  - **isNaN()**, que devuelve true si el parámetro que se le pasa no es NaN.

```
var myNumber = 2;
while (myNumber != Infinity) {
 myNumber = myNumber * myNumber;
}
```

```
isNaN(NaN); isNaN(100/"Apple"); // returns true
isNaN(10); isNaN("10"); // returns false
```

# Tipos de datos

No se aconseja utilizarlos porque ralentizan el programa y pueden producir efectos secundarios no deseados

- **Objeto Number:**

- Un número también puede ser de tipo Object

```
var x = 123;
var y = new Number(123);
```

```
// typeof x returns number
// typeof y returns object
```

```
var x = 500;
var y = new Number(500);
```

```
// (x == y) is true because x and y have equal values
```

```
var x = 500;
var y = new Number(500);
```

```
// (x === y) is false because x and y have different types
```

```
var x = new Number(500);
var y = new Number(500);
```

```
// (x == y) is false because objects cannot be compared
```

Tipos primitivos: paso por valor

```
var variable1 = 3;
var variable2 = variable1;

variable2 = variable2 + 5;
// Ahora variable2 = 8 y variable1 sigue valiendo 3
```

Tipos por referencia

```
// variable1 = 25 diciembre de 2009
var variable1 = new Date(2009, 11, 25);
// variable2 = 25 diciembre de 2009
var variable2 = variable1;

// variable2 = 31 diciembre de 2010
variable2.setFullYear(2010, 11, 31);
// Ahora variable1 también es 31 diciembre de 2010
```

# Tipos de datos

NOTA: new Date(y,m,d,h,min,s,ms).

-Como mínimo hay que incluir el año (y) y el mes (m).

-OJO, el parámetro m empieza con 0 para Enero, por lo tanto el 1 es febrero, y el -1 es el mes diciembre del año anterior.

-new Date() nos devuelve fecha actual

- **Number - propiedades:**

Property	Description
MAX_VALUE	Returns the largest number possible in JavaScript
MIN_VALUE	Returns the smallest number possible in JavaScript
NEGATIVE_INFINITY	Represents negative infinity (returned on overflow)
NaN	Represents a "Not-a-Number" value
POSITIVE_INFINITY	Represents infinity (returned on overflow)

- Ejemplo:

```
var x = Number.MAX_VALUE;
```

# Tipos de datos



- **Number - propiedades:**
  - Ejecuta el siguiente script y observa cuál es el número más grande representable por JavaScript, el número más cercano a 0 y el valor especial que representa el infinito

```
<script type="text/javascript">
 alert("Max value: " + Number.MAX_VALUE);
 alert("Min value: " + Number.MIN_VALUE);
 alert("Valor especial: " + Number.MAX_VALUE * 2);
</script>
```



# Tipos de datos

- **Number - métodos:**

Devuelve String

Method	Description
<b>toString()</b>	<pre>var myNumber = 128; myNumber.toString(16); // returns 80 myNumber.toString(2);</pre>
<b>toExponential()</b>	<pre>var x = 9.656; x.toExponential(2); // returns 9.66e+0</pre>
<b>toFixed()</b>	<pre>var x = 9.656; x.toFixed(0); // returns 10 x.toFixed(2); // returns 9.66</pre>
<b>toPrecision()</b>	<pre>var x = 9.656; x.toPrecision(); // returns 9.656 x.toPrecision(2); // returns 9.7</pre>
<b>valueOf()</b>	<pre>var x = 123; x.valueOf(); // returns 123 from variable x (123).valueOf(); // returns 123 from literal 123</pre>

# Tipos de datos

- **string:** Las cadenas son indexable y por ende iterables
  - Se pueden representar letras, dígitos, signos de puntuación o cualquier otro carácter de Unicode.
  - La cadena de caracteres se debe definir entre comillas dobles o comillas simples.

```
var carname = "Volvo XC60"; var answer = "It's alright";
var carname = 'Volvo XC60'; var answer = "He is called 'Johnny'";
 var answer = 'He is called "Johnny"';
```

- Cada carácter de la cadena se encuentra en una posición a la que se puede acceder individualmente. Siendo el primer carácter el de la posición 0.

# Tipos de datos

- **string:**

- Se puede conocer la longitud de una cadena con el método **length**.

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

- Para poder incluir cualquier carácter en las cadenas de caracteres, es necesario el uso de caracteres de escape.

Secuencia de escape	Resultado
\\	Barra invertida
\n	Salto de línea
\t	Tabulación horizontal
\v	Tabulación vertical
\f	Salto de página
\r	Retorno de carro
\b	Retroceso
\'	Comilla simple
\"	Comilla doble

# Tipos de datos

- **template literals**

Son literales de texto que habilitan el uso de expresiones incrustadas.

- Se usan para:

- Usar cadenas de texto de más de una línea
- Interpolación de cadenas de texto con ellas, es decir, otra forma de concatenar texto y variable sin usar el signo `+`.

```
`cadena de texto`
```

```
`línea 1 de la cadena de texto
línea 2 de la cadena de texto`
```

```
`cadena de texto ${expresión} texto`
```

Ojo Comillas invertidas!!!

# Tipos de datos

- **Objeto String:**
  - Podemos crear objetos tipo String.

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

// typeof x will return string
// typeof y will return object
```

No se aconseja utilizarlos porque  
ralentizan el programa y pueden  
producir efectos secundarios no deseados

# Tipos de datos

- **string- métodos:**

<code>charAt()</code>	Returns the character at the specified index (position)
<code>charCodeAt()</code>	Returns the Unicode of the character at the specified index
<code>concat()</code>	Joins two or more strings, and returns a copy of the joined strings
<code>fromCharCode()</code>	Converts Unicode values to characters
<code>indexOf()</code>	Returns the position of the first found occurrence of a specified value in a string
<code>lastIndexOf()</code>	Returns the position of the last found occurrence of a specified value in a string
<code>localeCompare()</code>	Compares two strings in the current locale
<code>match()</code>	Searches a string for a match against a regular expression, and returns the matches
<code>toUpperCase()</code>	Converts a string to uppercase letters
<code>trim()</code>	Removes whitespace from both ends of a string
<code>valueOf()</code>	Returns the primitive value of a String object

# Tipos de datos

- **string- métodos:**

`replace()`

Searches a string for a value and returns a new string with the value replaced

`search()`

Searches a string for a value and returns the position of the match

`slice()`

Extracts a part of a string and returns a new string

`split()`

Splits a string into an array of substrings

`substr()`

Extracts a part of a string from a start position through a number of characters

`substring()`

Extracts a part of a string between two specified positions

`toLocaleLowerCase()`

Converts a string to lowercase letters, according to the host's locale

`toLocaleUpperCase()`

Converts a string to uppercase letters, according to the host's locale

`toLowerCase()`

Converts a string to lowercase letters

`toString()`

Returns the value of a String object

# Tipos de datos

- **boolean** :
  - También conocido como valor lógico.
  - Sólo admite dos valores: `true` o `false`.
  - Es muy útil a la hora de evaluar expresiones lógicas o verificar condiciones.
  - Otros valores pueden tomar el valor de `true` o `false`:
    - `false`: `0`, `null`, `undefined`, `""`, `NaN`
    - `true`: `1` y básicamente cualquier otro valor que no esté en la lista de `false`.
  - Se recomienda declararlos con texto positivo:
    - `userIsLogged`



# Conversión entre tipos de datos

- Javascript es un lenguaje de programación no tipado.
- En ocasiones podemos convertir una variable de un tipo a otro (**typecasting**)
  - **toString()**: permite convertir variables de cualquier tipo a variables de cadena de texto (Método de Number).

Ejemplo:

```
var variable1 = true;
variable1.toString(); // devuelve "true" como cadena.
```

## Métodos globales

- **String()** : convierte el valor pasado en string
- **Boolean()**: convierte el valor pasado en un booleano

# Conversión entre tipos de datos

- **parseInt()** : convierte la variable que se le indica en un número entero
- **parseFloat()**, que convierte la variable que se le indica en un número decimal.

- **Number()**

```
x = true;
Number(x); // returns 1
x = false;
Number(x); // returns 0
x = new Date();
Number(x); // returns 1404568027739
x = "10"
Number(x); // returns 10
x = "10 20"
Number(x); // returns NaN
```

# Conversión entre tipos de datos

## Ejercicio



- Investiga más sobre la conversión de tipos en el tutorial de JS de w3schools.com:
  - [https://www.w3schools.com/js/js\\_type\\_conversion.asp](https://www.w3schools.com/js/js_type_conversion.asp)

# Ejercicio



- ¿Cuál será el resultado del siguiente código? ¿por qué?

```
alert (7 + 7 + 7);
```

```
alert (7 + 7 + "7");
```

```
alert ("7" + 7 + 7);
```

# ÍNDICE

1. Historia de JavaScript
2. Características de JavaScript
3. “Hola mundo” con JavaScript
4. El lenguaje JavaScript: sintaxis
5. Entrada y salida de datos
6. Variables
7. Tipos de datos
8. **Operadores**
9. Estructuras de control

# Operadores

- JavaScript utiliza principalmente cinco tipo de operadores:
  - Aritméticos.
  - Lógicos.
  - De asignación.
  - De comparación.
  - Condicionales.

# Operadores

- **Operadores aritméticos:**
  - Permiten realizar cálculos elementales entre variables numéricas.

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation ( <u>ES2016</u> )
/	Division
%	Modulus (Division Remainder)
++	Increment
--	Decrement

# Operadores

## Operadores de incremento y decremento:

```
var numero1 = 5;
var numero2 = 2;
numero3 = numero1++ + numero2;
```

-----

```
var numero1 = 5;
var numero2 = 2;
numero3 = ++numero1 + numero2;
```



# Operadores

## Operadores de incremento y decremento:

- Si el operador `++` se indica como **prefijo** del identificador de la variable, su valor se **incrementa antes** de realizar cualquier otra operación.
- Si el operador `++` se indica como **sufijo** del identificador de la variable, su valor se **incrementa después** de ejecutar la sentencia en la que aparece.

# Operadores

- **Operadores lógicos:**
  - Combinan diferentes expresiones lógicas con el fin de evaluar si el resultado de dicha combinación es verdadero o falso.

Operador	Nombre
& &	Y
	O
!	No

# Operadores

- **Operadores lógicos:**

- Ejecuta el siguiente script y observa su resultado



```
<script type="text/javascript">
 var op1 = prompt("Introduce el primer valor lógico (true o false): ", true);
 var op2 = prompt("Introduce el segundo valor lógico (true o false): ", true);
 alert("El resultado es: " + (op1 && op2));
</script>
```

# Operadores

- **Operadores lógicos:**

- **Not**

- Si la variable original contiene un **número**:
      - **false** → si el número es 0
      - **true** → en cualquier otro caso.
    - Si la variable original contiene una **cadena de texto**:
      - **false** → si la **cadena no contiene ningún carácter**
      - **true** → en cualquier otro caso

# Operadores

- **Operadores lógicos:**

- Not

Ejemplo:

```
var cantidad = 0;
vacio = !cantidad; // vacio = true
cantidad = 2;
vacio = !cantidad; // vacio = false
```

---

```
var mensaje='';
sinMensaje = !mensaje; // sinMensaje = true
mensaje = "hola mundo";
sinMensaje = !mensaje; // sinMensaje = false
```

# Operadores

- **Operadores de asignación:**

- Permiten obtener métodos abreviados para evitar escribir dos veces la variable que se encuentra a la izquierda del operador.

Operador	Nombre
<code>+=</code>	Suma y asigna
<code>-=</code>	Resta y asigna
<code>*=</code>	Multiplica y asigna
<code>/=</code>	Divide y asigna
<code>%=</code>	Módulo y asigna

Ejemplo:

```
var deudas= 1500;
```

```
deudas -= 300;
```

```
/*igual que escribir
deudas=deudas-300*/
```

# Operadores

- **Operadores de comparación:**
  - Permiten comparar todo tipo de variables y devuelve un valor booleano.

Operador	Nombre
<	Menor que
<=	Menor o igual que
==	Igual
>	Mayor que
>=	Mayor o igual que
!=	Diferente
===	Estrictamente igual
!==	Estrictamente diferente

# Operadores

## Operadores de comparación:

- Cuando se comparan **cadena de texto** con los operadores  $>$  y  $<$ , JavaScript compara letra a letra comenzando desde la izquierda hasta que se encuentre una diferencia entre las dos letras.
- Para determinar si una letra es mayor o menor que otra, se considera que:  $0 < 1 < 2 < \dots < 9 < A < B < \dots < Z < a < b < \dots < z$



# Operadores

## Operadores de comparación:

- El operador "idéntico" se indica mediante tres signos de igualdad (===) y devuelve true solamente si los dos operandos son exactamente iguales sin necesidad de realizar ninguna conversión.

```
10 == "10"; // devuelve true
```

```
0 == false; // devuelve true
```

```
"" == false //devuelve true, == convierte "" a 0
```

El resultado de estas comparaciones se debe a la conversión de datos, ante la comparación regular (==), los operandos primero se convierten a número.

El comparador estricto (===) evita esa conversión previa

```
10 === "10"; // devuelve false
```

# Operadores

- **Operadores condicionales:**

- Permite indicar al navegador que ejecute una acción en concreto después de evaluar una expresión.

Operador	Nombre
<code>?:</code>	Condicional

- Si la expresión antes del operador es verdadera, se utiliza el primer valor a la derecha. En caso contrario se utiliza el segundo valor a la derecha.

```
test ? expression1 : expression2
```

# Operadores

- **Operadores condicionales:**

- Ejemplo:

```
<script type="text/javascript">
 var dividendo = prompt("Introduce el dividendo: ");
 var divisor = prompt("Introduce el divisor: ");
 (divisor != 0) ? alert(dividendo/divisor) : alert("No es posible la división
por cero");
</script>
```

# Operadores

- **Operadores condicionales:**

- Ejemplo:

```
var now = new Date();
var greeting = "Good" + ((now.getHours() > 17) ? " evening." : " day.");
```

- Equivalente con if ... else

```
var now = new Date();
var greeting = "Good";
if (now.getHours() > 17)
 greeting += " evening.";
else
 greeting += " day.";
```

# Operadores



- **Operadores condicionales:**

- Ejercicio

- Crea un documento que le pida al usuario un año y nos devuelva si es o no bisiesto. Utiliza el operador ?:
- Crea un documento que pruebe el uso de este operador de forma anidada. ¿funciona?

# Orden de preferencia de operadores

- Negación (!) / Incremento (++) / Decremento (--)
- Multiplicación (\*) / División (/) / Resto (%)
- Suma (+) / Resta (-)
- Relacionales mayor - menor (>, <, >=, <=)
- Igualdad (==) / Desigualdad (!=)
- And lógico (&&)
- Or lógico (||)
- Asignación (=, +=, -=...)

# ÍNDICE

1. Historia de JavaScript
2. Características de JavaScript
3. “Hola mundo” con JavaScript
4. El lenguaje JavaScript: sintaxis
5. Entrada y salida de datos
6. Variables
7. Tipos de datos
8. Operadores
9. **Estructuras de control**

# Estructuras de control

- Permiten modificar el flujo de ejecución de las instrucciones de la aplicación.
- Vamos a dividir estas estructuras en dos tipos:
  - De selección
    - if
    - switch
  - Iterativas
    - while
    - for



# Estructuras de selección

- Con las sentencias condicionales se puede gestionar la toma de decisiones y el posterior resultado por parte del navegador.
- Dichas sentencias evalúan condiciones y ejecutan ciertas instrucciones en base al resultado de la condición.
- Las estructuras de selección en JavaScript son:
  - `if`: Permite redirigir un curso de acción según la evaluación de una condición
  - `switch`: De acuerdo con el valor de una variable, permite ejecutar un grupo u otro de sentencias

# Estructuras de selección

- `if` – sintaxis (1):

```
if (expresión) {
 instrucciones
}
```

# Estructuras de selección

- `if` – sintaxis (2):

```
if (expresión) {
 instrucciones_si_true
} else {
 instrucciones_si_false
}
```

# Estructuras de selección

- `if` – sintaxis (3):

```
if (expresión_1) {
 instrucciones_1
} else if (expresión_2) {
 instrucciones_2
} else {
 instrucciones_3
}
```

# Estructuras de selección

- `switch` – sintaxis:

```
switch (expresión){
 case valor1:
 instrucciones a ejecutar si expresión
 break;
 case valor2:
 instrucciones a ejecutar si expresión
 break;
 case valor3:
 instrucciones a ejecutar si expresión
 break;
 default:
 instrucciones a ejecutar si expresión es
diferente a
 los valores anteriores
}
```

# Estructuras de selección

- `switch` – sintaxis (rangos):

```
switch (expresión){
 case valor1:
 case valor2:
 case valor3:
 instrucciones a ejecutar
 break;
 default:
 instrucciones a ejecutar si
 expresión es diferente a los
 valores anteriores o no hay
 break
}
```

```
switch (true) {
 case (valor>X1 && valor <X2):
 instrucciones a ejecutar
 break;
 case (valor>X3 && valor <X4):
 instrucciones a ejecutar
 break;
 default:
 instrucciones a ejecutar si
 expresión es diferente a los
 valores anteriores o no hay
 break
}
```

# Estructuras iterativas

- Las estructuras de control iterativas o de repetición, **inician o repiten un bloque de instrucciones si se cumple una condición o mientras se cumple una condición.**
- Las estructuras de iterativas en JavaScript son:
  - **switch:** Ejecuta un grupo de sentencias solo cuando se cumpla una condición
  - **for:** Ejecuta un grupo de sentencias un número determinado de veces. JS tiene bastante particularidades con este tipo de bucle, destacaremos:

Tipos especiales  
de bucles "for"

- for in
- for of

# Estructuras iterativas

- **while – sintaxis:**

(1)

```
while (expresión) {
 instrucciones
}
```

(2)

```
do {
 instrucciones
} while (expresión)
```

}  
Forma especial del bucle "while"  
que garantiza que el contenido del  
bucle se ejecute al menos una vez.



# Estructuras iterativas

- `while` – sintaxis:

Básicamente, una estructura `while` necesita que se le indiquen 3 cosas:

- Cuándo empezar
- Cuándo parar
- Cómo obtener el siguiente item, es decir, el paso

```
var start=0 //cuando empezar
while (start <10) { //cuando parar
 console.log(start);
 start=start + 2; //paso
}
```

# Estructuras iterativas

- **for – sintaxis:**

```
for(v_inicial; exp_condicional; inc_o_dec_variable) {
 cuerpo_del_bucle
}
```

# Estructuras iterativas

- `for` (buena práctica)
  - Sobre un objeto o array es ineficiente debido al acceso, en cada iteración, a la propiedad
  - Se puede optimizar con la declaración única de variable y evitando el operador `++`:

```
var i=0,
 max,
 miArray=[];

....
for (i = 0, max = miArray.length; i < max; i = i + 1){
 //procesar elementos
}
```

# Estructuras iterativas

- **for / in** — Se utiliza para desplazarnos por las propiedades de un objeto

- **sintaxis:**

```
for(var in object){
 cuerpo_del_bucle
}
```

```
var person = {fname:"John", lname:"Doe", age:25};
```

```
var text = "";
```

```
var x;
```

```
for (x in person) {
 text += person[x];
}
```



John Doe 25

Obtenemos los valores de las propiedades del objeto sin necesidad de conocer las claves (fname, lname y age)

# Estructuras iterativas

- `for / in` (buena práctica):
  - Con este tipo de bucle accedemos a las propiedades o métodos que no pertenecen al objeto sino a su prototipo.
  - Para evitarlo se debe usar el método `hasOwnProperty()` que devuelve `true` si la propiedad es del objeto y no de su prototipo.  
El objeto tiene la propiedad especificada

```
for (propiedad in miObjeto){
 if (miObjeto.hasOwnProperty(propiedad))
 //procesar elementos miObjeto[propiedad]
}
```

# Estructuras iterativas – ES6 novedades

## For ... of

Sirve para iterar arrays, objetos, ... al igual que `for..in` pero la diferencia es que `for..in` incluye los `key` (en Arrays los índices de posición) y `for..of` itera sobre los valores del objeto"

```
let list = [4, 5, 6];

for (let i in list) {
 console.log(i); // "0", "1", "2",
}

for (let i of list) {
 console.log(i); // "4", "5", "6"
}
```

- Para usar `for... of` el objeto debe ser iterable, es decir, definir un iterador (arrays, map, set). No se puede usar en objetos definidos por el usuario.

# Estructuras iterativas – ES6 novedades

- **For ... of**
  - En objetos como Map y Set **for..of** permite acceder a los valores almacenados.

```
let pets = new Set(["Cat", "Dog", "Hamster"]);
pets["species"] = "mammals";

for (let pet in pets) {
 console.log(pet); // "species"
}

for (let pet of pets) {
 console.log(pet); // "Cat", "Dog", "Hamster"
}
```

# Conversión de ES6 a ES5

```
1 let list = [4, 5, 6];
2
3 for (let i of list) {
4 console.log(i); // "4", "5", "6"
5 }
6
```

```
1 "use strict";
2
3 var list = [4, 5, 6];
4
5 var _iteratorNormalCompletion = true;
6 var _didIteratorError = false;
7 var _iteratorError = undefined;
8
9 try {
10 for (var _iterator = list[Symbol.iterator](), _step; !
 (_iteratorNormalCompletion = (_step = _iterator.next()).done);
 _iteratorNormalCompletion = true) {
11 var i = _step.value;
12
13 console.log(i); // "4", "5", "6"
14 }
15 } catch (err) {
16 _didIteratorError = true;
17 _iteratorError = err;
18 } finally {
19 try {
20 if (!_iteratorNormalCompletion && _iterator.return) {
21 _iterator.return();
22 }
23 } finally {
24 if (_didIteratorError) {
25 throw _iteratorError;
26 }
27 }
28 }
```



# Sentencias break y continue

- Las sentencias break y continue permiten manipular el comportamiento normal de los bucles, para detener el bucle o para saltarse algunas repeticiones.
  - **break:** Se usa para salir o saltar fuera de un bucle finalizándolo.
  - **continue:** Se usa para saltar hasta la siguiente iteración del bucle; permite saltar una o más iteraciones.

# Sentencias break y continue

- Si el programa llega a una instrucción de tipo **break**;, sale inmediatamente del bucle y continúa ejecutando el resto de instrucciones que se encuentran fuera del bucle.

```
var cadena = "En un lugar de la Mancha de cuyo nombre no quiero acordarme...";
var letras = cadena.split(""); //sin espacio entre comillas, array de caracteres
var resultado = "";
for (i in letras) {
 if (letras[i] == 'a') {
 break;
 }
 resultado += letras[i];
}

alert(resultado); // muestra "En un lug"
```

# Sentencias break y continue

- La utilidad de **continue** es que **permite** utilizar el bucle for para **filtrar** los **resultados en función de algunas condiciones** o cuando el valor de alguna variable coincide con un valor determinado.

```
var cadena = "En un lugar de la Mancha de cuyo nombre no quiero acordarme...";
var letras = cadena.split(""); var resultado = "";
for (i in letras) {
 if (letras[i] == 'a' {
 continue;
 }
 resultado += letras[i];
}
alert(resultado);
// muestra "En un lugar de l Mnch de cuyo nombre no quiero cordrme..."
```

# Sentencias break y continue

<b>CONTINUE</b>
<i>El bucle terminara cuando muestre el 10</i>
<pre>&lt;html&gt; &lt;head&gt;   &lt;script&gt;     function bucle()     {       var cont=1;       while(cont&lt;=10)       {         alert(cont);         cont++;         if(cont==5)           continue;       }     }   &lt;/script&gt; &lt;/head&gt; &lt;body onLoad=bucle();&gt;&lt;/body&gt; &lt;/html&gt;</pre>

<b>BREAK</b>
<i>El bucle terminara cuando muestre el 4</i>
<pre>&lt;html&gt; &lt;head&gt;   &lt;script&gt;     function bucle()     {       var cont=1;       while(cont&lt;=10)       {         alert(cont);         cont++;         if(cont==5)           break;       }     }   &lt;/script&gt; &lt;/head&gt; &lt;body    onLoad=bucle();&gt;&lt;/body&gt; &lt;/html&gt;</pre>

# ANEXOS

---

# JavaScript – Guía de estilo y convenciones

## • Convenciones para nombres

- Variables y nombres de funciones → camelCase
- Variables globales y constantes → MAYÚSCULA
- Constructores → la primera letra en mayúsculas (new)
- Métodos privados → comenzar con \_

No usar guiones  
No utilizar \$ como  
primer carácter

## • Poner **espacios** alrededor de los operadores y después de las ,

## • “Indentar” el código

- Usar siempre 4 espacios ( el espacio de las tabulaciones cambian entre los editores)

# JavaScript – Guía de estilo y convenciones

- **Reglas para las sentencias:**
  - Terminar con un `;` una sentencia simple

```
var values = ["Volvo", "Saab", "Fiat"];
```

```
var person = {
 firstName: "John",
 lastName: "Doe",
 age: 50,
 eyeColor: "blue"
};
```

- Para sentencias complejas (*funciones, bucles, condicionales*):
  - Poner el carácter de apertura `{` al final de la primera línea y precedido de un espacio.
  - Poner el carácter de cierre `}` en una nueva línea sin `;`

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

# JavaScript – Guía de estilo y convenciones

- **Reglas para los objetos:** ←

```
var person = {
 firstName: "John",
 lastName: "Doe",
 age: 50,
 eyeColor: "blue"
};
```

```
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

- **Para mayor legibilidad, evitar líneas con más de 80 caracteres**

```
document.getElementById("demo").innerHTML =
 "Hello Dolly.";
```



# JavaScript – Guía de estilo y convenciones

- **Simplificar la etiqueta `<script>`** `<script src="myscript.js">`
- **Extensiones** para los ficheros:
  - .html / .css / .js
- **Escribir el nombre de los ficheros en minúscula**

# JavaScript – Buenas prácticas

- Evitar variables globales
- Declarar siempre las variables locales
- Declarar las variables al inicio del script
- Inicializar las variables
- No declarar nunca un número, una cadena de caracteres o un booleano como Object

//Atención: antipatrón

```
function miFunc(a, b){
 resultado = a + b;
 return resultado;
}
```

//Atención: antipatrón

```
function miFunc() {
 var a = b = 0;
 //
}
```

```
function laFuncion(){
 var una_var = 1,
 otra_var = 5,
 el_nombre = "Santiago",
 unobjeto = {},
 i = 0;
 // Aquí iría el código
}
```

Patrón de  
declaración único

# JavaScript – Buenas prácticas

- **Patrón de espacio de nombres**

- Para evitar la contaminación del ámbito global con variables, se recomienda asignar un identificador y definir la aplicación como si fuese un objeto con propiedades y métodos:

```
var miAplicacion = {};
miAplicacion.estudiante = {
 "nombre": "Santiago",
 "apellido": "Alonso"
};
miAplicacion.curso = {
 "codigo": "C01",
 "titulo": "HTML5"
};
```

Formas de comprobar su existencia previa:

```
if (typeof miAplicacion === "undefined")
 var miAplicacion = {}
```

O en una sola línea:

```
var miAplicacion = miAplicacion || {};
```

# JavaScript – Buenas prácticas

- Usar `{}` en lugar de `new Object()`
- Usar `""` en lugar de `new String()`
- Usar `0` en lugar de `new Number()`
- Usar `false` en lugar de `new Boolean()`
- Usar `[]` en lugar de `new Array()`
- Usar `/()/` en lugar de `new RegExp()`
- Usar `function () {}` en lugar de `new function()`

```
var x1 = {};
var x2 = "";
var x3 = 0;
var x4 = false;
var x5 = [];
var x6 = /()/;
var x7 = function(){};
```

# JavaScript – Buenas prácticas

- Cuidado con las conversiones de tipo automáticas

```
var x = "Hello"; // typeof x is a string
x = 5; // changes typeof x to a number
```

- Usar el comparador ===

0 == "";	// true	0 === "";	// false
1 == "1";	// true	1 === "1";	// false
1 == true;	// true	1 === true;	// false



- Usar valores predefinidos

```
function myFunction(x, y) {
 if (y === undefined) {
 y = 0;
 }
}
```

- Finalizar un switch con un valor por defecto

# JavaScript – Errores comunes

- Utilizar el operador `==` para comparar el valor y el operador `===` para comparar el valor y el tipo de dato
- En la estructura switch se utiliza comparación estricta

```
var x = 10;
switch(x) {
 case "10": alert("Hello");
}
```

*Handwritten notes:* `int` with a red arrow pointing to `10`, and `STR` with a red arrow pointing to `"10"`.

¡¡No muestra nada!!



- No confundir suma con concatenación

```
var x = 10 + 5; // the result in x is 15
var x = 10 + "5"; // the result in x is "105"
```

# JavaScript – Errores comunes

- Confusiones con float

```
var x = 0.1;
var y = 0.2;
var z = x + y
if (z == 0.3)
```



0.30000000000000004

Errores de Redondeo

```
var z = (x * 10 + y * 10) / 10;
```

// z will be 0.3

- Una sentencia se puede partir en dos líneas, pero no en medio de un string

```
var x = "Hello
World!";
```

¡Error!

```
var x = "Hello \
World!";
```

# JavaScript – Errores comunes

- No partir en dos líneas la sentencia de retorno de una función ni poner la { en una línea distinta a return

```
function myFunction(a) {
 var
 power = 10;
 return
 a * power;
}
```

```
//Atención: antipatrón
function miFuncion(){
 return
 {
 nombre: "Santiago"
 };
}
console.log (typeof miFuncion()); //undefined
```

- JavaScript no permite índices nombrados en los arrays

```
var person = [];
person["firstName"] = "John";
person["lastName"] = "Doe";
person["age"] = 46;
```



# JavaScript – Errores comunes

- No finalizar la definición de un objeto o un array con una ,

```
points = [40, 100, 1, 5, 25, 10,]; person = {firstName:"John", lastName:"Doe", age:46,}
```

- **Null** → para objetos

Para que un objeto sea null ha tenido que ser definido, en caso contrario es undefined

```
if (typeof myObj !== "undefined" && myObj !== null)
```

- **Undefined** → para variables, propiedades y métodos

- En JavaScript no crea un nuevo ámbito para cada bloque de código

```
for (var i = 0; i < 10; i++) {
 // some code
}
return i;
```

➔ 10

# JavaScript – Errores comunes

```
var baz;
console.log(baz); //baz === undefined

var foo = 100;
var foo;
console.log(foo); //foo === 100
```

```
console.log(unaVar); //unaVar === undefined
console.log(otraVar); //otraVar === undefined

if (!unaVar) {
 var unaVar = true;
}
var otraVar = 'otra variable';

console.log(unaVar); //unaVar === true
console.log(otraVar); //otraVar === 'otra variable'
```



```
var unaVar, otraVar;
console.log(unaVar); //unaVar === undefined
console.log(otraVar); //otraVar === undefined

if (!unaVar) {
 unaVar = true;
}
otraVar = 'otra variable';

console.log(unaVar); //unaVar === true
console.log(otraVar); //otraVar === 'otra variable'
```



# JavaScript – Errores comunes

```
function miFuncionA(){
 var unaVar;
 for(var i=0; i<100; i++){
 unaVar = 'algun valor';
 }

 console.log(unaVar);
}
```

```
function miFuncionB(){
 for(var i=0; i<100; i++){
 var unaVar = 'algun valor';
 }

 console.log(unaVar);
}
```

# JavaScript – “use strict”

IE from version 10.  
Firefox from version 4.  
Chrome from version 13  
Safari from version 5.1  
Opera from version 12

- **Declaración de variables**

- Si “use strict” se incluye al principio del fichero tendrá un ámbito global y afectará a todo el código

```
"use strict";
x = 3.14;
```

¡**Error!**

```
"use strict";
myFunction();
```

```
function myFunction() {
 y = 3.14; // This will also cause an error
}
```

- Si se declara dentro de una función, sí tendrá ámbito local

```
x = 3.14; // This will not cause an error.
myFunction();
```

```
function myFunction() {
 "use strict";
 y = 3.14; // This will cause an error
}
```

# JavaScript – “use strict”

- **Características**

- Es un modo de escribir código seguro
- No se pueden crear variables o parámetros repetidos
- Usar alguna propiedad no inicializada o algún objeto o variable no existen reportará un error.
- No se pueden eliminar variables o propiedades con `delete`
- No se puede utilizar: `octal` ni la barra de escape, `with()`, `eval()`, palabras reservadas de futuro
- El valor de `this` no referencia al objeto global si está sin definir.
- <https://www.youtube.com/watch?v=Bs01JKyg9Qo>

# JavaScript – Manejo de Errores

- **Try** → permite determinar los bloques de código sobre los que se van a captar los errores durante la ejecución.
- **Catch** → permite manejar los errores detectados con try

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

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

```
<script>
```

```
try {
```

```
 adddler("Welcome guest!");
```

```
}
```

```
catch(err) {
```

```
 document.getElementById("demo").innerHTML = err.message;
```

```
}
```

```
</script>
```

```
</body>
```

```
</html>
```



err.name  
err.message

//Se pone adddler para forzar el error

# JavaScript – Manejo de Errores

- **Throw** → permite crear mensajes de error personalizados

```
<script>
function myFunction() {
 var message, x;
 message = document.getElementById("message");
 message.innerHTML = "";
 x = document.getElementById("demo").value;
 try {
 if(x == "") throw "empty";
 if(isNaN(x)) throw "not a number";
 x = Number(x);
 if(x < 5) throw "too low";
 if(x > 10) throw "too high";
 }
 catch(err) {
 message.innerHTML = "Input is " + err;
 }
}
</script>
```

# JavaScript – Manejo de Errores

- **finally** → se indica después de try y catch. Aquí se indica el código que queremos que se ejecute independiente de lo que haya pasado.

```
function myFunction() {
 var message, x;
 message = document.getElementById("message");
 message.innerHTML = "";
 x = document.getElementById("demo").value;
 try {
 if(x == "") throw "is empty";
 if(isNaN(x)) throw "is not a number";
 x = Number(x);
 if(x > 10) throw "is too high";
 if(x < 5) throw "is too low";
 }
 catch(err) {
 message.innerHTML = "Error: " + err + ".";
 }
 finally {
 document.getElementById("demo").value = "";
 }
}
```



# JavaScript – Manejo de Errores

- Podemos encontrar los siguientes tipos de errores:

Error Name	Description
EvalError	An error has occurred in the eval() function
RangeError	A number "out of range" has occurred
ReferenceError	An illegal reference has occurred
SyntaxError	A syntax error has occurred
TypeError	A type error has occurred
URIError	An error in encodeURIComponent() has occurred

# JavaScript – eval()

- Ejecuta el argumento que se le pasa como cadena de texto
- Éste puede ser una expresión o varias sentencias
- Devuelve el valor de la última expresión evaluada

```
var x = 5;
```

```
var str = "if (x == 5) {alert('z is 42'); z = 42;} else z = 0; ";
```

```
document.write("<p>z es ", eval(str));
```



z es 42

```
<script>
function myFunction() {
 var x = 10;
 var y = 20;
 var a = eval("x * y") + "
";
 var b = eval("2 + 2") + "
";
 var c = eval("x + 17") + "
";
 document.write(a + b + c);
}
</script>
```



200

4

27

[https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos\\_globales/eval](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/eval)


# JavaScript – ES6 novedades

- **Template strings**

- Nos permite interpolar strings de una forma más sencilla

```
//ES5
var nombre1 = 'Javascript';
var nombre2 = 'awesome';
console.log(nombre1 + " is " + nombre2);

//ES6
var nombre1 = 'Javascript';
var nombre2 = 'awesome';
console.log(`${nombre1} is ${nombre2}`);
```



# JavaScript – ES6 novedades

- **Strings multilínea**

- Podemos escribir strings multilínea sin utilizar el operador +.

```
//ES5
var saludo = "Hola
 mundo";
```

```
//ES6


var saludo = `Hola
 mundo`;

console.log(`Hola
 mundo`);
```

```
//ES5
var saludo = "Hola"+
 "mundo";
```

```
//ES6
var saludo = `Hola
mundo`;

console.log(`Hola
mundo`);
```



# JavaScript – ES6 novedades

- **Literales octales , binarias y hexadecimales**
  - Podemos crear literales octales y binarias con los prefijos (0b), (0o) y (0x) respectivamente.

```
var a = 0b11110111; //binario
console.log(a); //503
var b = 0o767; //octal
console.log(b); //503
var c = 0x1f7; // hexadecimal
console.log(c); //503
```