

SOC

Servei d'Ocupació  
de Catalunya



Generalitat  
de Catalunya



Unió Europea  
Fons social europeu  
L'FSE inverteix en el teu futur



## MÓDULO 1. MF0951\_2

INTEGRAR COMPONENTES SOFTWARE EN PÁGINAS WEB

### UNIDAD FORMATIVA 1.

UF1305 INTEGRAR COMPONENTES SOFTWARE EN PÁGINAS WEB.



Formació



Formació  
online



Gestió de  
Bonificacions



Desenvolupament  
de Persones



Selecció



Tecnologies de  
la Informació



Millora  
Contínua



Servei de  
Prevenció Aliè





## 2 Javascript. Introducció

### 2.0 ¿Qué es Javascript?

#### 2.1 Características del lenguaje.

\_Descripción del lenguaje orientado a eventos.

\_Descripción del lenguaje interpretado.

#### 2.2 Relación del lenguaje de guión y el lenguaje de marcas.

\_Extensión de las capacidades del lenguaje de marcas.

\_Adición de propiedades interactivas.

\_¿Qué hace a JavaScript único?



## 2.3

### Tipos de scripts

- \_Inmediatos
- \_Diferidos
- \_\_Diferidos async, defer
- \_Híbridos
- \_y dinámicos
- \_Script en el Head
- \_Script en fichero externo





## 2.4

### Sintaxis del lenguaje de gui3n.

- \_Etiquetas identificativas dentro del lenguaje de marcas.
- \_Especificaciones y caracterfsticas de las instrucciones.
- \_Consola de desarrollador
- \_Elementos del lenguaje de gui3n.
- \_\_Variables.
- \_\_Palabras reservadas
- \_\_Operaciones.
- \_\_Comparaciones.
- \_\_Asignaciones.
- \_Objetos del lenguaje de gui3n.
- \_\_M3todos.
- \_\_Eventos.
- \_\_Atributos.
- \_\_Funciones.



## 2.5

### Ejecución de un script

- \_Ejecución al cargar la página.
- \_Ejecución después de producirse un evento.
- \_Ejecución del procedimiento dentro de la página.
- \_Tiempos de ejecución.
- \_Errores de ejecución.

## 2.6

### Ejercicios

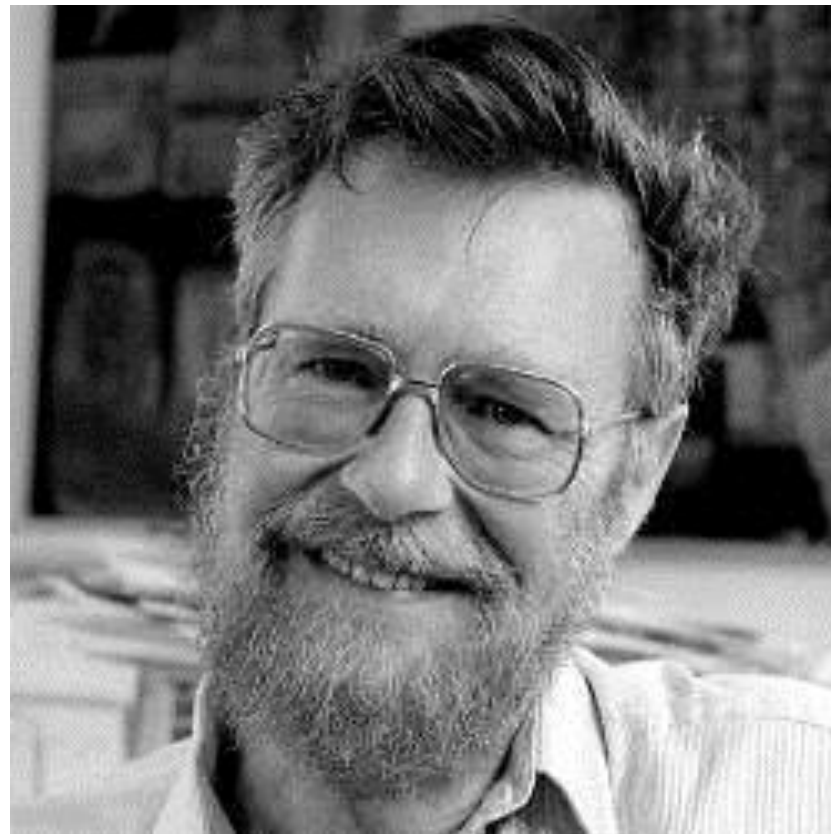
Desvincular ficheros HTML y JS

Fichero HTML con errores de ubicación de scripts



"El uso de COBOL daña la mente. Su enseñanza debería ser considerada como un ataque criminal"

-- E. W. Dijkstra



Edsger Wybe Dijkstra fue un científico de la computación de los Países Bajos. Poco después de su muerte en el 2002, recibió la distinción ACM PODC Influential Paper Award en computación distribuida por su trabajo en la auto-estabilización en programas computacionales.

# JavaScript.Introducción



# Javascript

## Características del lenguaje



## 2.1 \_¿Qué es Javascript?

- JavaScript fue creado para “**dar vida a las páginas web**”.
- Los programas en este lenguaje **se llaman scripts**.
- Se pueden escribir **directamente en el HTML** de una página web y ejecutarse automáticamente a medida que se carga la página.
- Los scripts se proporcionan y ejecutan **como texto plano**.
- **No necesitan** preparación especial o **compilación** para correr.
- En este aspecto, **JavaScript** es **muy diferente** a otro lenguaje llamado **Java**.

## 2.1 \_¿Qué es Javascript?

### ¿Por qué se llama JavaScript?

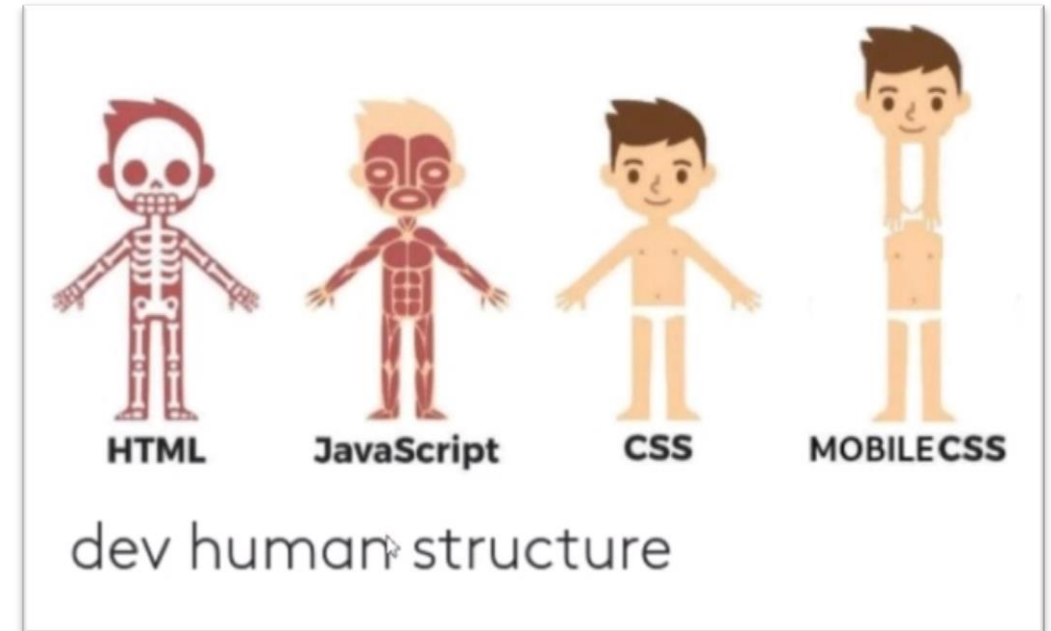
Cuando JavaScript fue creado, **inicialmente** tenía otro nombre: “**LiveScript**”. Pero Java era muy popular en ese momento, así que se decidió que el posicionamiento de un nuevo lenguaje como un “Hermano menor” de Java ayudaría.

Pero a medida que evolucionaba, JavaScript se convirtió en un lenguaje completamente independiente con su **propia especificación** llamada **ECMAScript**, y ahora no tiene ninguna relación con Java.

## 2.1 \_Características del lenguaje.

### \_Descripción del lenguaje orientado a eventos.

- JavaScript es un **lenguaje de programación o de secuencias de comandos**
- Implementa **funciones complejas en páginas web**
- Es la **tercera capa del pastel** de las tecnologías web estándar, dos de las cuales (HTML y CSS)
- El HTML se utiliza para conformar el esqueleto y la estructura de los contenidos de una página web.
- El CSS define el estilo y la apariencia web.
- Javascript **rompe con lo estático** del HTML y **permite crear elementos dinámicos e interactivos**, mejorando ampliamente la interacción de los usuarios con una página web.



## 2.1 \_Características del lenguaje.



### \_Descripción del lenguaje orientado a eventos.

```
<p>Player 1: Chris</p>
```

```
p {  
  font-family: 'helvetica neue', helvetica, sans-serif;  
  letter-spacing: 1px;  
  text-transform: uppercase;  
  text-align: center;  
  border: 2px solid rgba(0,0,200,0.6);  
  background: rgba(0,0,200,0.3);  
  color: rgba(0,0,200,0.6);  
  box-shadow: 1px 1px 2px rgba(0,0,200,0.4);  
  border-radius: 10px;  
  padding: 3px 10px;  
  display: inline-block;  
  cursor: pointer;  
}
```

```
const para = document.querySelector('p');  
para.addEventListener('click', updateName);
```

```
function updateName() {  
  let name = prompt('Enter a new name');  
  para.textContent = 'Player 1: ' + name;  
}
```

## 2.1 \_Características del lenguaje.

- **Lenguaje del lado del cliente:** se ejecuta en la máquina del propio cliente a través de un navegador. javascript, HTML, CSS
- **Lenguaje orientado a objetos:** es un lenguaje orientado a objetos.
- **De tipado débil o no tipado:** no es necesario especificar el tipo de dato al declarar una variable.
- **De alto nivel:** su sintaxis es fácilmente comprensible por su similitud al lenguaje de las personas. su sintaxis se encuentra alejada del nivel máquina.
- **Lenguaje interpretado:** permite convertir las líneas de código en el lenguaje de la máquina. Esto tiene un gran número de ventajas como la reducción del procesamiento en servidores web al ejecutarse directamente en el navegador del usuario.



## 2.2 Relación del lenguaje de guión y el lenguaje de marcas..



- **Etiquetas o tags:** cada una de ellas se indica encerrada entre corchetes angulares, de forma que para el inicio de la etiqueta se marca `<xxx>` y para el cierre de etiqueta se añade una barra lateral `</xxx>`.
- **Elementos:** éstos definen la estructura del documento HTML y van indicando al navegador como debe presentar el contenido. Los elementos están compuestos por una etiqueta de inicio, el contenido y una etiqueta de cierre. También existen elementos vacíos que no tienen contenido, este tipo no posee etiqueta de cierre y su etiqueta de inicio acaba con `" />"`.
- **Atributos:** indican las propiedades asociadas a cada elemento, tienen una estructura `nombreAtributo=valorAtributo`, se ubican dentro de la etiqueta de inicio de cada elemento.



## 2.2 Relación del lenguaje de guión y el lenguaje de marcas..

```
<HTML>

  <HEAD>

    <TITLE>

      Título que queramos darle a la página Web.

    </HEAD>

  <BODY>

    Información que queramos ofrecer al usuario

  </BODY>

</HTML>
```

En esta estructura es donde se va a añadir el lenguaje de guión, las etiquetas que aparecen son:

- <HTML> y </HTML>, para indicar el principio y el fin del documento HTML.
- <HEAD> y </HEAD>, indica el encabezado y la descripción del documento.
- <TITLE> y </TITLE>, se corresponde con el título que aparece en la barra del navegador que esté utilizando el usuario.
- <BODY> y </BODY>, indica el cuerpo del documento, en el gráfico hemos indicado que contiene la información que queramos ofrecer al usuario.

## 2.2 Relación del lenguaje de guión y el lenguaje de marcas..

### \_Extensión de las capacidades del lenguaje de marcas.

- JavaScript: su principal función es facilitar la interacción de la página Web en función de los eventos que el usuario produzca o la propia página Web.
- HTML: se utiliza para dar estructura a la página Web, a través de HTML se puede definir listas, formularios, párrafos, imágenes, etc..
- CSS (Cascading Styles Sheets): se utiliza ahora aplicar estilos visuales a la página, indica al navegador cómo se debe mostrar los diferentes elementos que se han agregado con HTML.

## 2.2 Relación del lenguaje de guión y el lenguaje de marcas..

\_Adicción de propiedades interactivas.

Por ejemplo, en el navegador JavaScript es capaz de:

- **Agregar nuevo HTML a la página, cambiar el contenido existente y modificar estilos.**
- **Reaccionar a las acciones del usuario**, ejecutarse con los clics del ratón, movimientos del puntero y al oprimir teclas.
- Enviar **solicitudes de red** a servidores remotos, descargar y cargar archivos (Tecnologías llamadas [AJAX](#) y [COMET](#)).
- **Obtener y configurar cookies**, hacer preguntas al visitante y mostrar mensajes.
- **Recordar datos en el lado del cliente** con el almacenamiento local (“**local storage**”).

## 2.2 Relación del lenguaje de guión y el lenguaje de marcas..

\_Adicción de propiedades interactivas.

### Ejemplos de restricciones :

- JavaScript en el navegador **no puede leer y escribir arbitrariamente archivos en el disco duro**, copiarlos o ejecutar programas.
- No tiene acceso directo a funciones del Sistema operativo (OS).
- **Diferentes pestañas y ventanas generalmente no se conocen entre sí.** A veces sí lo hacen, por ejemplo, cuando una ventana usa JavaScript para abrir otra. Pero incluso en este caso, JavaScript no puede acceder a la otra si provienen de diferentes sitios (de diferente dominio, protocolo o puerto).
- JavaScript puede fácilmente comunicarse a través de la red con el servidor de donde la página actual proviene. Pero su capacidad para recibir información de otros sitios y dominios esta bloqueada..

## 2.2 Relación del lenguaje de guión y el lenguaje de marcas..

\_Adicción de propiedades interactivas.

# SI

Mostrar la fecha y la hora actual del usuario, crear relojes animados, calcular la edad de una persona y personalizar otras acciones relacionadas con la fecha.

Calcular pagos de préstamos e hipotecas y otros datos financieros.

Averiguar no sólo el nombre del navegador del usuario, sino también su número de versión y el sistema operativo con el que se está ejecutando.

Trabajar con ventanas del navegador adicionales, lo que significa que podemos abrirlas, redimensionarlas, cerrarlas e, incluso, cambiar el contenido a voluntad.

Enviar al navegador del usuario a otra página.

Configurar una página protegida mediante contraseña.

Crear un "árbol" de navegación que haga más fácil para los usuarios moverse a través de las páginas del sitio.

Validar los valores de un formulario antes de enviarlo al servidor. Por ejemplo, podemos asegurarnos de que ciertos campos se han rellenado.

Crear un "carro de la compra" que almacenará los objetos que un usuario ha seleccionado para comprar.

Crear asignaciones sofisticadas, como hacer que fluya texto a lo largo de la pantalla o configurar una exhibición de imágenes con efectos de transición.

## 2.2 Relación del lenguaje de guión y el lenguaje de marcas..

\_Adicción de propiedades interactivas.

# NO

Escribir datos permanentemente en un archivo. Por ejemplo, no podremos tomar los datos de un libro de visitas y añadirlos a una página que muestre los mensajes.

Acceder a archivos del servidor.

Acceder a archivos de la computadora del usuario (excepto a un archivo especial denominado cookie).

Recabar información sobre el usuario, incluyendo la dirección de correo electrónico y la dirección IP.

Realizar compras basadas en tarjetas de crédito para autorización y pago.

Crear juegos de múltiples jugadores.

Obtener datos de una base de datos del servidor.

Gestionar la transmisión de archivos al servidor.

# Javascript

## Tipos de scripts





**JAVASCRIPT** es un lenguaje de programación que se utiliza para mejorar la presentación de nuestras páginas HTML.

- No tiene nada que ver con JAVA.
- No necesita compilación.
- Es un lenguaje interpretado.

## 2.3 \_Tipos de scripts: **inmediatos**, diferidos e híbridos.

El código puede ser de tres tipos: inmediato – diferidos – híbridos ..... Y dinámicos.

**Scripts inmediatos**: que se ejecutan **nada más cargar la página** y van dentro del **Body**.

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

  <script type="text/javascript">
    document.write("Este es un mensaje escrito con JS dentro del <body> de la página.");
  </script>
</body>
</html>
```

**Scripts diferidos (async, defer):** que se cargan con la página pero no se ejecutan hasta que el usuario hace algo (pulsar un botón, una tecla, etc). Van dentro del **Body**.



En los sitios web modernos **los scripts suelen ser más “pesados” que el HTML**, el tamaño de la descarga es grande y el **tiempo de procesamiento es mayor**.

- Cuando el navegador carga el HTML y se encuentra con una etiqueta `<script>...</script>`, no puede continuar construyendo el DOM. Debe ejecutar el script en el momento.

```
<p>...contenido previo al script...</p>
<script src="https://javascript.info/article/script-async-defer/long.js?speed=1"></script>
<!-- Esto no es visible hasta que el script sea cargado -->
<p>...contenido posterior al script...</p>
```

Lo mismo sucede con los **scripts externos** `<script src="..."></script>`, el navegador tiene que **esperar** hasta que el script sea descargado, ejecutarlo y solo después procesa el resto de la página.

## 2.3 \_Tipos de scripts: inmediatos, **diferidos** e híbridos.



### 2 importantes problemas:

- Los **scripts** no pueden ver los elementos del DOM que se encuentran debajo de él por lo que **no pueden agregar controladores de eventos**, etc.
- Si hay un **script muy pesado** en la parte superior de la página, este **“bloquea la página”**. Los usuarios no pueden ver el contenido de la página hasta que sea descargado y ejecutado.

## 2.3 \_Tipos de scripts: inmediatos, **diferidos** e híbridos.



### Scripts diferidos (async, defer)

Por ejemplo podemos poner el **script en la parte inferior de la página** por lo que podrá ver los elementos sobre él y no bloqueará la visualización del contenido de la página.

**Solución NO perfecta.** Por ejemplo el navegador solo se dará cuenta del script (y podrá empezar a descargarlo) **después de descargar todo el documento HTML.**

Para documentos **HTML extensos** eso puede ser **un retraso notable.**

## Scripts diferidos (async, defer)

**defer**

El atributo defer indica al navegador que **no espera por el script**. En lugar de ello, debe **seguir procesando el HTML**, construir el DOM. El **script carga “en segundo plano”** y se ejecuta cuando el **DOM esta completo**.

```
<p>...contenido previo script...</p>  
<script defer src="https://javascript.info/article/script-async-defer/long.js?speed=1"></script>  
<!-- Inmediatamete visible -->  
<p>...contenido posterior al script...</p>
```

El atributo defer es solo para scripts externos  
El atributo defer es ignorado si el <script> no tiene el atributo src.

## 2.3 \_Tipos de scripts: inmediatos, **diferidos** e híbridos.

### Scripts diferidos (async, **defer**)



#### **defer**

El atributo defer indica al navegador que **no espere por el script**. En lugar de ello, debe **seguir procesando el HTML**, construir el DOM. El script carga “en segundo plano” y se ejecuta cuando el DOM esta completo.

```
<p>...contenido previo script...</p>
<script defer src="https://javascript.info/article/script-async-defer/long.js?speed=1"></script>
<!-- Inmediatamete visible -->
<p>...contenido posterior al script...</p>
```

El atributo **defer** es solo para **scripts externos**

El atributo **defer** es ignorado si el **<script>** no tiene el atributo **src**.



## 2.3 \_Tipos de scripts: inmediatos, **diferidos** e híbridos.

### Scripts diferidos (async, **defer**)



#### **defer**

En otras palabras:

- Los scripts con defer **nunca bloquean la página**.
- Los scripts con defer **siempre se ejecutan cuando el DOM esta listo (pero antes del evento **DOMContentLoaded**)**.

El **DOMContentLoaded** evento se activa cuando el documento HTML inicial se ha cargado y analizado por completo, sin esperar a que las hojas de estilo, las imágenes y los submarcos terminen de cargarse.

```
<p>...contenido previo a los scripts...</p>
<script>
  document.addEventListener('DOMContentLoaded', () => alert("¡DOM listo después del defer!"));
</script>
<script defer src="https://javascript.info/article/script-async-defer/long.js?speed=1"></script>
<p>...contenido posterior a los scripts...</p>
```

El contenido de la página se muestra inmediatamente.

DOMContentLoaded espera por el script diferido. Solo se dispara cuando el script es descargado y ejecutado.

## 2.3 \_Tipos de scripts: inmediatos, **diferidos** e híbridos.

### Scripts diferidos (**async**, defer)



#### **async**

El atributo async es de alguna manera **como defer**. También hace el **script no bloqueante**. Pero tiene importantes **diferencias** de comportamiento.

En otras palabras, los scripts async cargan en segundo plano y se ejecutan cuando están listos.

El DOM y otros scripts no esperan por ellos, y ellos no esperan por nada.

Un script totalmente independiente que se ejecuta en cuanto se ha cargado.

Scripts diferidos (**async**, defer)**async**

Los unos no esperan por lo otros. El que cargue primero (probablemente small.js), se ejecuta primero.

```
<p>...contenido previo a los scripts...</p>
<script> document.addEventListener('DOMContentLoaded', () => alert("¡DOM listo!"));
</script>
<script async src="https://javascript.info/article/script-async-defer/long.js">
</script> <script async src="https://javascript.info/article/script-async-defer/small.js"></script>
<p>...contenido posterior a los scripts...</p>
```

Los scripts asincrónicos son excelentes cuando incluimos **scripts de terceros** (contadores, anuncios, etc) en la página debido a que **ellos no dependen de nuestros scripts y nuestros scripts no deberían esperar por ellos**.

```
<!-- Google Analytics is usually added like this -->
<script async src="https://google-analytics.com/analytics.js"></script>
```

**El atributo async es solo para scripts externos**

Tal como defer, el atributo async se ignora si la etiqueta <script> no tiene src.

**Scripts híbridos**: se definen tanto en el **Head** como en el **Body**.

Lo más normal es colocar los scripts Javascript justo antes del cierre de la etiqueta **<body>**

```
[... contenido de la página ...]  
<script src="mis-scripts.js"></script>  
</body>
```

## 2.3 \_Tipos de scripts: inmediatos, diferidos e híbridos. **y dinámicos**



**Scripts Dinámicos:** El script comienza a cargar tan pronto como es agregado al documento (\*).

```
let script = document.createElement('script');  
script.src = "/article/script-async-defer/long.js";  
document.body.append(script); // (*)
```

**Los scripts dinámicos se comportan como async por defecto**

- Ellos no esperan a nadie y nadie espera por ellos.
- El script que carga primero se ejecuta primero (**load-first order**)

## 2.3 \_Tipos de scripts: inmediatos, diferidos e híbridos. y dinámicos



### Resumen

Ambos, async y defer, tienen algo **en común**: la descarga de tales **scripts no bloquean el renderizado de la página**.

Por lo **cual el usuario puede leer el contenido de la página y familiarizarse con la página inmediatamente**.

## 2.3 \_Tipos de scripts: inmediatos, diferidos e híbridos. y dinámicos



### Resumen

Pero hay algunas diferencias esenciales entre ellos:

**async**

#### Orden

Load-first order.

#### DOMContentLoaded

El orden del documento no importa.

El que carga primero ejecuta primero

Irrelevante. Puede cargar y ejecutarse mientras el documento no ha sido completamente descargado, eso puede pasar si el script es pequeño o está en cache y el documento es suficientemente extenso.

**defer**

Document order

(como en el documento).

Ejecutan después de que el documento es cargado y analizado (espera si es necesario), justo antes de DOMContentLoaded.



## 2.3 \_Tipos de scripts: inmediatos, diferidos e híbridos. y dinámicos



### Resumen

En la práctica, **defer** es usado para **scripts** que necesitan todo el **DOM** y/o si su **orden de ejecución relativa es importante**.

Y **async** es usado para **scripts independientes**, como contadores y anuncios donde el orden de ejecución no importa.

## 2.3 \_Tipos de scripts: inmediatos, diferidos e híbridos. y dinámicos



**La página sin scripts debe ser utilizable**

Ten en cuenta: si usas **defer** o **async**, el usuario verá la página antes de que el script sea cargado.

En tal caso algunos componentes gráficos probablemente no estén listos.

No olvides poner alguna **señal de “cargando”** y **deshabilitar los botones** que aún no estén funcionando.

Esto permite al usuario **ver claramente qué puede hacer en la página y qué está listo y qué no.**

## 2.3 \_Tipos de scripts: inmediatos, diferidos e híbridos.

### Script en el head del HTML



```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Document</title>
  <script type="text/javascript">
    alert("Hola mundo – 1 !");
    alert("Hola mundo – 2 !");
    alert("Hola mundo – 3 !");
    alert("Hola mundo – 4 !");

  </script>
</head>
<body>
  <h1>Curso de Javascript</h1>
  <p>Mi primer Hola mundo!</p>
</body>
</html>
```

Index\_1

¿Dónde?

- Head
- Al final del body , cuando haya todo el árbol de etiquetas

```
<script type="text/javascript">
</script>
```

./codigo/index.html

mensaje

Esta página dice  
Hola mundo-1 !

Aceptar

## 2.3 \_Tipos de scripts: inmediatos, diferidos e híbridos.

### Script en fichero externo al HTML



```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <script type="text/Javascript" src="../codigo/assets/js/hola-mundo.js"></script>
</head>
<body>
  <h1>Curso de Javascript</h1>
  <p>Mi primer Hola mundo!</p>
</body>
</html>
```

Index\_2

#### Fichero externo

- Crear subcarpeta /assets/js
- Crer fichero "hola-mundo.js"
- Copiar script

```
<script type="text/Javascript"
src="fichero.js">
</script>
```

```
alert("Hola mundo-1 !");
document.write('Hola mundo! ');
```

# Javascript

## Sintaxis



## 2.4 \_Sintaxis de Javascript. Etiquetas identificativas dentro del lenguaje de marcas.



¿Cuántas etiquetas conocéis?



## 2.4 \_Sintaxis de Javascript. Características de las instrucciones

- **No se tienen en cuenta los espacios en blanco y las nuevas líneas:** como sucede con XHTML, el intérprete de JavaScript ignora cualquier espacio en blanco sobrante, por lo que el código se puede ordenar de forma adecuada para entenderlo mejor ([tabulando las líneas, añadiendo espacios, creando nuevas líneas, etc.](#))
- **Se distinguen las mayúsculas y minúsculas:** al igual que sucede con la sintaxis de las etiquetas y elementos XHTML. Sin embargo, si en una página XHTML se utilizan indistintamente mayúsculas y minúsculas, la página se visualiza correctamente, siendo el único problema la no validación de la página. En cambio, [si en JavaScript se intercambian mayúsculas y minúsculas el script no funciona.](#)

## 2.4 \_Sintaxis de Javascript. Características de las instrucciones

- **No se define el tipo de las variables:** al crear una variable, **no es necesario indicar el tipo de dato que almacenará**. De esta forma, una misma variable puede almacenar diferentes tipos de datos durante la ejecución del script.
- **No es necesario terminar cada sentencia con el carácter de punto y coma (;):** en la mayoría de lenguajes de programación, es obligatorio terminar cada sentencia con el carácter ;. Aunque JavaScript no obliga a hacerlo, **es conveniente seguir la tradición** de terminar cada sentencia con el carácter del punto y coma (;).
- **Se pueden incluir comentarios:** los **comentarios se utilizan para añadir información en el código** fuente del programa. Aunque el contenido de los comentarios no se visualiza por pantalla, si que se envía al navegador del usuario junto con el resto del script, por lo que es necesario extremar las precauciones sobre la información incluida en los comentarios.

```
//VARIABLES  
//Contenedor de información
```

```
var pais = "España";  
var continente = "Europa";
```



## 2.4 \_Sintaxis de Javascript. Consola del navegador

El código es propenso a **errores**. Es muy probable que cometas errores ...  
Oh, ¿de qué estoy hablando? Definitivamente vas a cometer errores, al menos si eres un humano, no un robot.

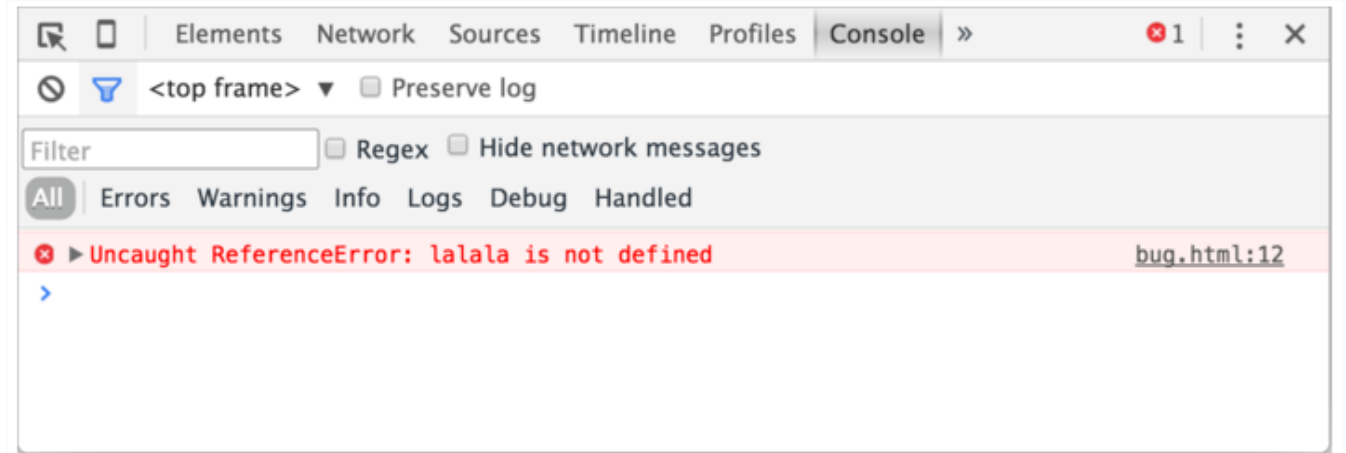
Para ver los errores y obtener mucha otra información útil sobre los scripts, se han incorporado “**herramientas de desarrollo**” en los navegadores.



## 2.4 \_Sintaxis de Javascript. Consola del navegador



Abre la página [bug.html](#).  
Hay un **error en el código JavaScript dentro de la página**.  
Está oculto al visitante regular  
**Presione F12**  
Se ve algo así:



El aspecto exacto de las herramientas de desarrollador depende de su versión de Chrome.  
Aquí podemos ver el mensaje de **error de color rojo**.  
En este caso, el script contiene un comando desconocido “lalala”.  
A la derecha, hay un enlace con la **fuentes bug.html:12** con el número de línea del error

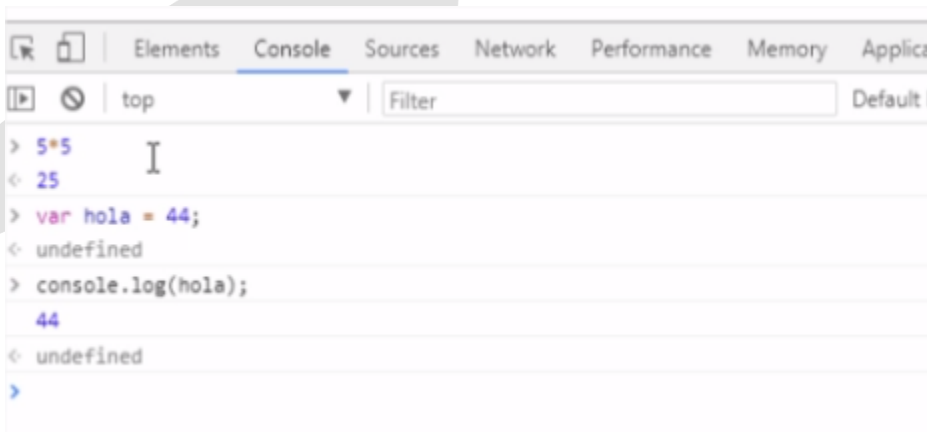


Opciones -> Más herramientas -> Herramientas desarrolladores

[F12]

- Elements
- Console (javascript, debugear, ...)

- **Elements** HTML
- **Sources**: ver archivos
- **Networks**: petición Ajax a backend ver peticiones



```

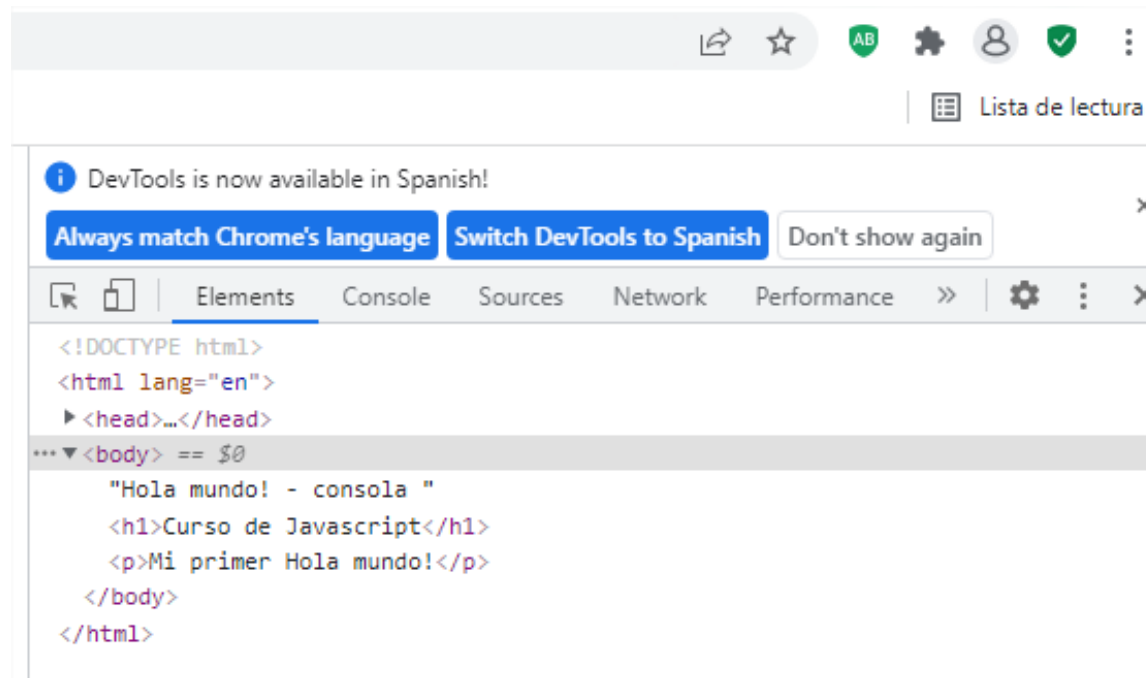
> 5*5
< 25
> var hola = 44;
< undefined
> console.log(hola);
44
< undefined
>

```

```

alert("Hola mundo-1 !");
document.write("Hola mundo! ");
console.log("Muestra esto en la consola.");
console.log(88+4);

```





```
//VARIABLES
```

```
//Contenedor de información
```

```
var pais = "España";  
var continente = "Europa";  
var antigüedad = 2022;  
var pais_y_continente = pais + ' ' + continente;
```

```
console.log(pais, continente, antigüedad);  
alert (pais_y_continente);
```

```
pais = "Andorra";  
continente = "Latam";
```

```
console.log(pais, continente, antigüedad); //Andorra Latam 2022  
alert (pais_y_continente); //España y Europa
```

Además, el tipo de dato puede cambiar a lo largo de la ejecución.

Así, una variable inicializada con un 0 será un número, pero si más adelante se le asigna una cadena, su tipo cambiará.

Tiene su ventaja pero también sus inconvenientes

**variable1=1**



**variable1="Pedro"**



### LISTADO DE PALABRAS RESERVADAS ECMA SCRIPT 6

Break	case	class	catch	const	continue	debugger	default	delete
	do	else	export	extends	finally	for	function	if
	in	instanceof		let	new	return	super	switch
	throw	try	typeof	var	void	while	with (en-US)	yield

**Fuente:** <https://developer.mozilla.org>

### OTRAS PALABRAS RESERVADAS POR DIVERSOS MOTIVOS

Enum	implements		package	protected		static	interface	private	public
	abstract	boolean	byte	char	double	final	float	goto	int
	native	short	synchronized		transient	volatile			long

### Operaciones aritméticas

Las operaciones aritméticas son aquellas que nos permiten el trabajo con números tales como la suma, la resta, etc. Estas operaciones aritméticas se realizarán utilizando operadores: los operadores más comunes son el de suma (+), el de resta (-), el de multiplicación (\*) y el de división (/). Por ejemplo:

```
<script>
a = 3 - 5 // a valdrá -2
a = 3; b = 4 // a valdrá 3 y b 4
c = a + b // c valdrá 7
</script>
```

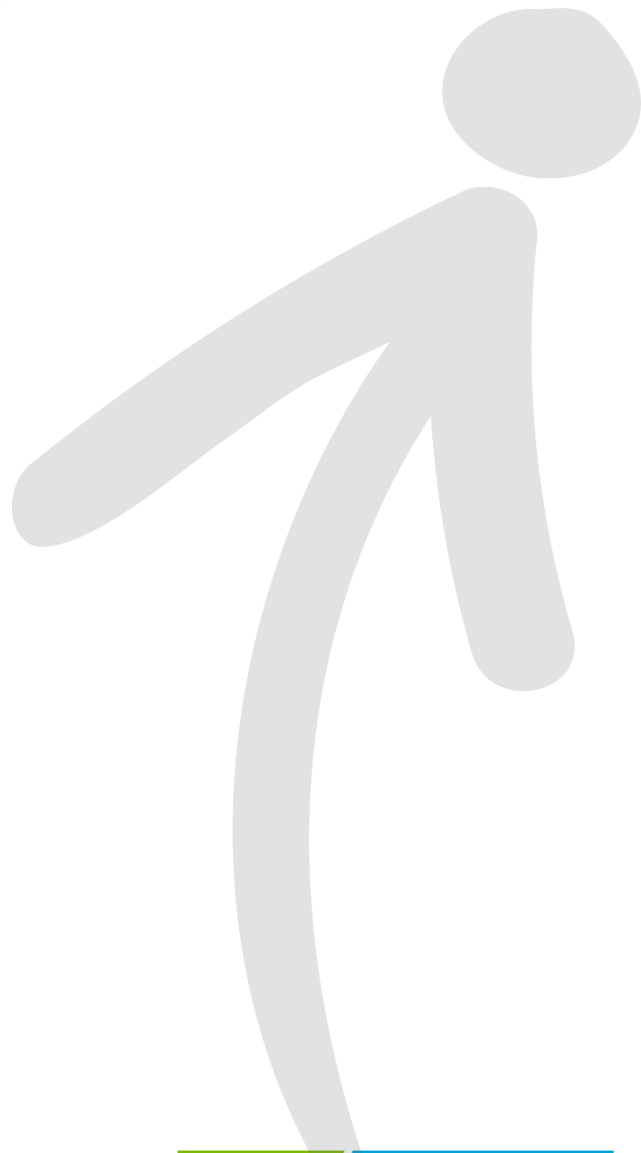
```
<script>
a = 4 // a valdrá 4
a++ // a valdrá 5
a-- // a volverá a valer 4
</script>
```

```
<script>
a = "java"
b = "script"
c = a+b // c valdrá "javascript"
</script>
```

### Operaciones lógicas

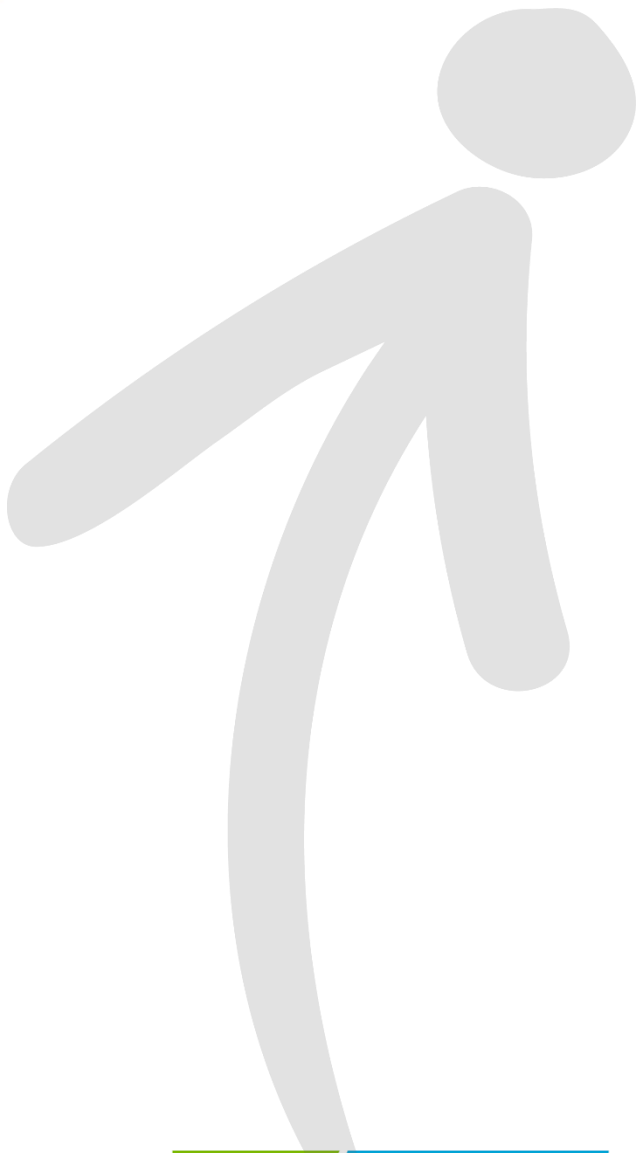
Nos permiten complicar las expresiones a evaluar, insertando más de una comparación en la misma expresión. Los operadores de los que disponemos son AND (&&), que devuelve true si se cumplen ambas comparaciones, OR (||) que lo hace si se cumple alguna y NOT (!) que niega una expresión.

```
<script>
a = ((1 == 1) && (2 == 5)) // a valdrá false
</script>
```



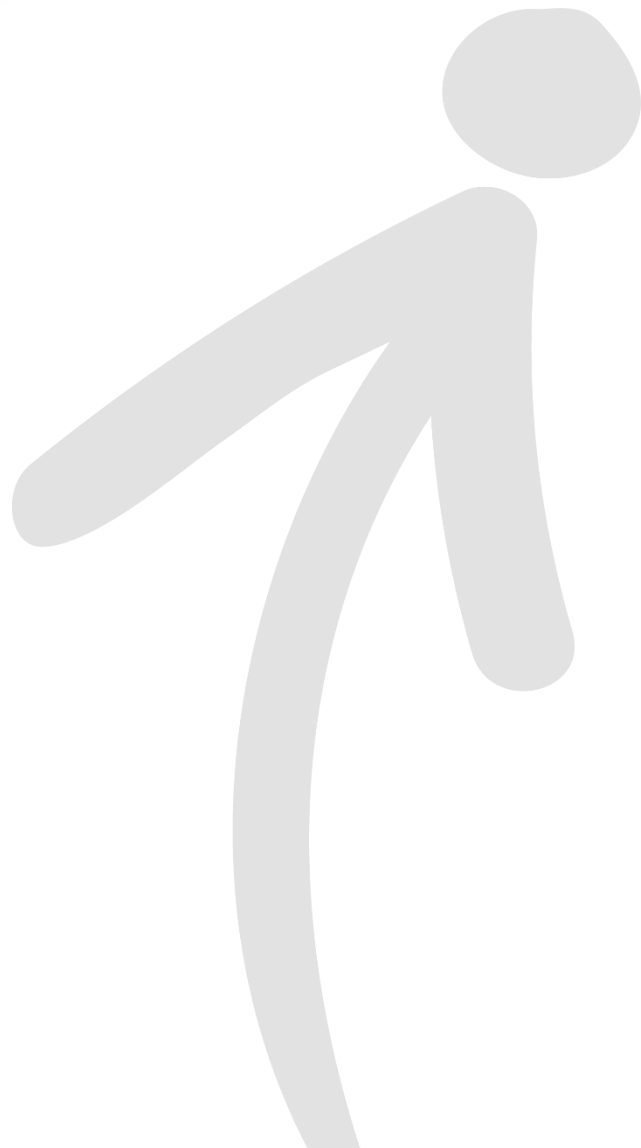
Operador	Descripción	Ejemplos que devuelven verdadero <sup>1</sup>
Igual (==)	Devuelve true si los operandos son iguales.	3 == var1 "3" == var1 3 == '3'
Distinto (!=)	Devuelve true si los operandos no son iguales.	var1 != 4 var2 != "3"
Igual estricto (===)	Devuelve true si los operandos son iguales y del mismo tipo.	3 === var1
Distinto estricto (!===)	Devuelve true si los operandos no son iguales y/o no son del mismo tipo.	var1 !== "3" 3 !== '3'
Mayor que (>)	Devuelve true si el operando izquierdo es mayor que el derecho.	var2 > var1 "12" > 2
Mayor o igual que (>=)	Devuelve true si el operando izquierdo es mayor o igual que el derecho.	var2 >= var1 var1 >= 3
Menor que (<)	Devuelve true si el operando izquierdo es menor que el derecho.	var1 < var2 "12" < "2"
Menor o igual que (<=)	Devuelve true si el operando izquierdo es menor o igual que el derecho.	var1 <= var2 var2 <= 5





Operador de asignación	Acción
=	Asignación. Asigna la parte de la derecha del igual a la parte de la izquierda. A la derecha se colocan los valores finales y a la izquierda generalmente se coloca una variable donde queremos guardar el dato.
+=	Asignación con suma. Realiza la suma de la parte de la derecha con la de la izquierda y guarda el resultado en la parte de la izquierda.
-=	Asignación con resta.
*=	Asignación de la multiplicación.
/=	Asignación de la división
%=	Se obtiene el resto y se asigna.





ahorros = 7000	//asigna un 7000 a la variable ahorros
ahorros += 3500	//incrementa en 3500 la variable ahorros, ahora vale 10500
ahorros /= 2	//divide entre 2 mis ahorros, ahora quedan 5250

Partiendo del concepto de que JavaScript es un lenguaje en el cual se pueden desarrollar aplicaciones basadas en el paradigma de programación orientada a objetos, conocido por las siglas POO.

En este paradigma, POO, cada objeto es una entidad independiente que puede recibir y enviar mensajes desde o hacia objetos del sistema.

Los conceptos fundamentales que se deben tener claros:

- **Clase:** Determina las características de los objetos. Por ejemplo, si contamos con una clase que se denomina *Moto* que permita crear varios objetos del tipo *Moto*.
- **Objeto:** Se define como una instancia de una clase, si continuamos con el ejemplo anterior, crearemos diferentes objetos dentro de la clase *Moto* como podría ser: *Moto Harley Davidson*, *Moto Suzuki*, *Moto Yamaha*, etc..
- **Atributos:** Definen las características de los objetos, siguiendo con el ejemplo, se pueden añadir atributos como el color, longitud, modelo, etc..
- **Método:** Define la capacidad de los objetos, es decir, operaciones que pueden realizar los objetos, siguiendo con el ejemplo, podrían ser métodos el cambio de marchas, arrancar, etc.

Estas acciones, por lo general, pueden clasificarse en las siguientes categorías:

### **Simular una acción del usuario.**

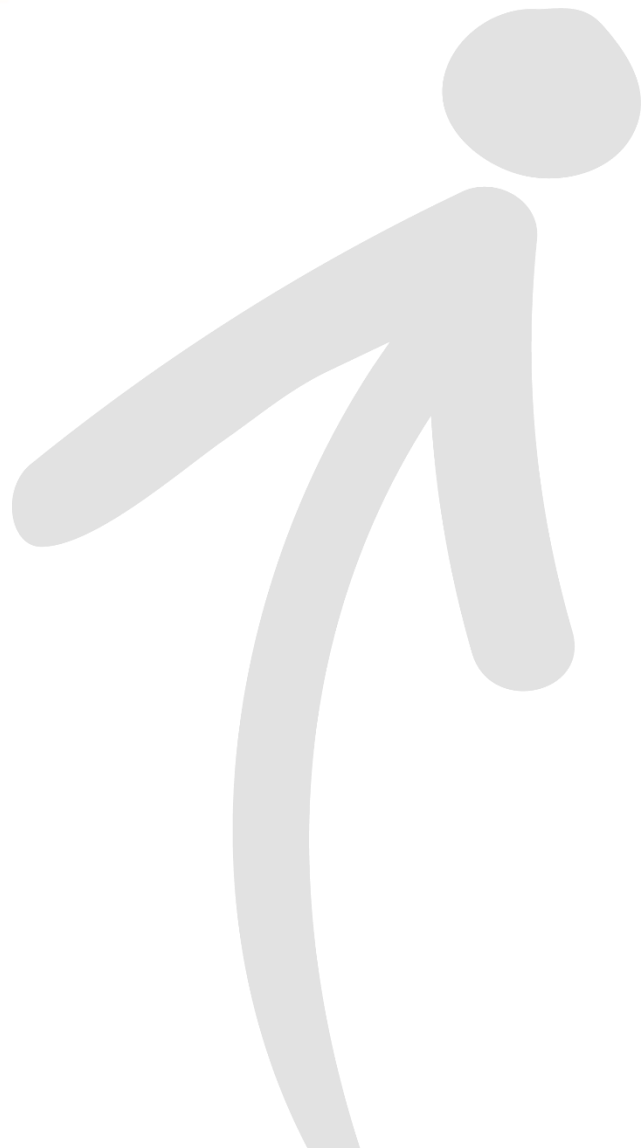
Como por ejemplo, vemos que el método `submit()` del objeto `Form` envía un formulario al servidor, de igual forma que cuando un usuario hace clic en el botón `Submit` de cualquier formulario.

### **Realizar un cálculo.**

Por ejemplo, el método `sqrt()` del objeto `Math` calcula la raíz cuadrada de un número.

### **Manipulación de un objeto.**

Por ejemplo, el método `toLowerCase()` del objeto `String` modifica todas las letras de la cadena escribiéndolas en minúsculas.



Un evento es una acción que realiza el usuario sobre algún elemento de nuestra página al interactuar con él, como por ejemplo pasar el puntero del ratón por encima de una imagen o hacer clic sobre un botón.

Gracias a los eventos, la interactividad de nuestra página aumenta considerablemente ya que podremos hacer que reaccione ante las acciones del usuario.

Esta característica es especialmente explotada por DHTML (Dynamic HTML) para crear variedad de efectos y situaciones.

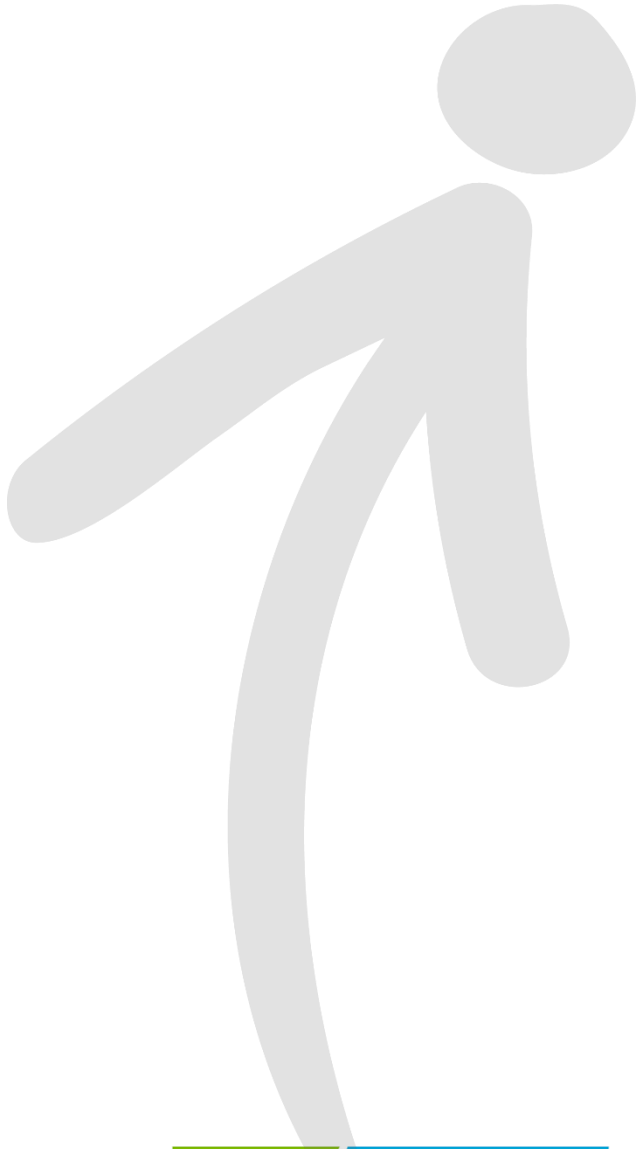
JavaScript es capaz de detectar estos eventos y a la vez nos permite asociarles unas instrucciones que se ejecutarán cuando se produzcan.

Un evento como tal carece de utilidad por sí mismo, así que se hace necesario asociarles una función o código JavaScript que se ejecutará cuando se produzca dicho evento.

Los atributos son las características que describen a cada uno de los objetos, éstos se definen en la clase, son similares al conjunto de variables que componen a cada objeto. Algunos de los atributos de eventos que podemos encontrar en JavaScript cuando están insertados como un atributo HTML del documento son los siguientes:

onclick
ondblclick
onmousedown
onmouseup
onmouseover
onmousemove
onmouseout

onkeypress
onkeydown
onkeyup
onfocus
onblur
onload
onunload
onchange
onselect



```
<script type="text/javascript">

//Función que muestra un mensaje

Function mostrarMensaje() {

    Alert("Hola");

}

//Asignación del manejador

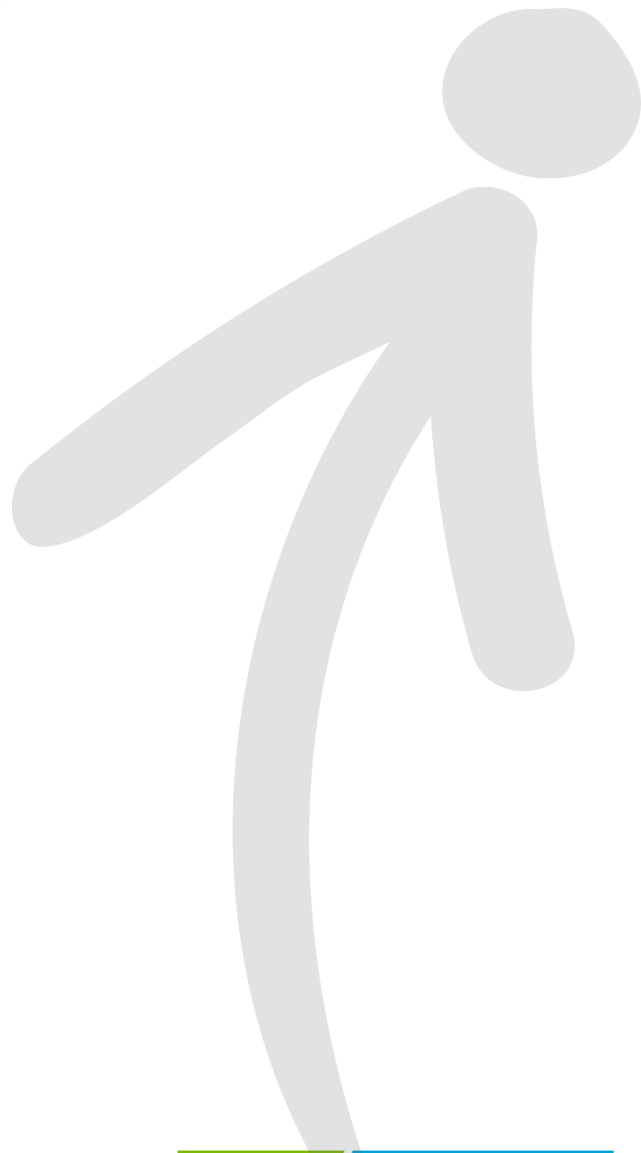
Objeto.onClick = mostrarMensaje;

Objeto.onclick = function mostrarMensaje() {

    Alert("Hola mundo");

}

</script>
```



Es muy común a la hora de programar que nos surja la necesidad de ejecutar un conjunto de acciones de forma habitual o simplemente nos convenga que se ejecuten de forma independiente para dar mayor claridad al código.

Pues bien, la solución a este problema son las funciones, que se encargan de agrupar una serie de acciones dentro de un mismo bloque para ejecutarlas cuando y cuantas veces queramos.

De esta forma, podemos tener por ejemplo una función que se encargue siempre de sumar dos números, con lo que nos evitaremos repetir esa suma en distintas partes del código.



Importante

El nombre de los parámetros de las funciones no podrán coincidir con el de ninguna de las variables declaradas dentro de la función.

**2.5.**

# JavaScript

## Ejecución de un script





## 2.5 Ejecución de un script. \_Al cargar la página

Tres maneras que conozco de cargar una función de JavaScript una vez se haya cargado la pagina.

1.- Poniendo el tag onload en el <body> de nuestra página para que ejecute nuestra función:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
</head>
  <title>Test</title>
  <script>
    function miFuncion() {
      alert('OK');
    }
  </script>
</head>
<body onload="miFuncion();">
  mi pagina...
</body>
</html>
```

## 2.5 Ejecución de un script. \_Al cargar la página

Tres maneras que conozco de cargar una función de JavaScript una vez se haya cargado la pagina.

2.- Llamando a la función directamente desde javascript con window.onload:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <head>
    <title>Test</title>
    <script>
      function miFuncion() {
        alert('OK');
      }
      window.onload=miFuncion;
    </script>
  </head>
  <body>
    mi pagina...
  </body>
</html>
```

## 2.5 Ejecución de un script. \_Al cargar la página

Tres maneras que conozco de cargar una función de JavaScript una vez se haya cargado la pagina.

3.- Ejecutar nuestro código dentro del window.onload sin llamar a ninguna función:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
</head>
  <title>Test</title>
  <script>
    window.onload=function() {
      alert('OK');
    }
  </script>
</head>
<body>
  mi pagina...
</body>
</html>
```

## 2.5 Ejecución de un script. \_Después de un evento

```
const myFunction = (option) => {  
  switch (option) {  
    case 1:  
      console.log('- Option 1');  
      // code ...  
      break;  
    case 2:  
      console.log('- Option 2');  
      // code ...  
      break;  
    default:  
      break;  
  }  
};
```

```
<button onclick="myFunction(1)">Option 1</button>  
<button onclick="myFunction(2)">Option 2</button>
```

```
<a href="javascript:finestraSecundaria('https://www.twooweb.com')">Diseño Web</a>

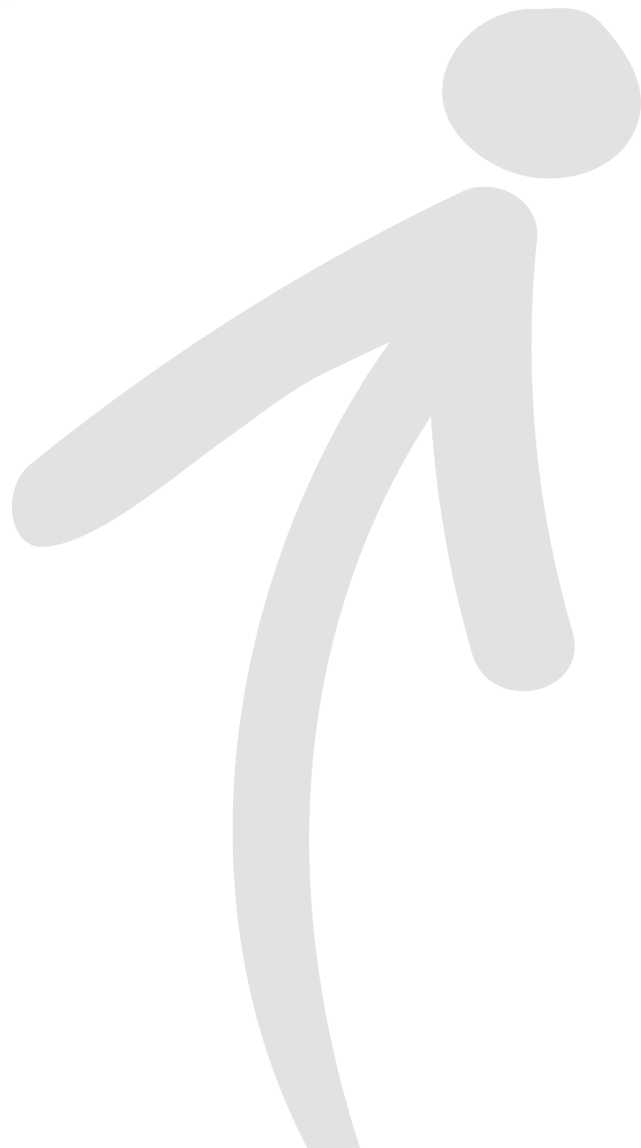
<script language=javascript>
    function finestraSecundaria (url){
        window.open(url, "Diseño Web", "width=300, height=200")
    }
</script>
```

**2.6.**

# JavaScript

## Ejercicios





Desvincular ficheros HTML y JS  
Fichero HTML con errores de ubicación de scripts

ejercicio 231

ejercicio 232

[Las 89 preguntas más importantes sobre Javascript](#)



### INSTRUCCIONES BÁSICAS PARA FAMILIARIZARSE CON JAVASCRIPT

Instrucciones básicas que sirven para empezar a practicar con scripts sencillos.  
Concretamente me estoy refiriendo a:

```
window.prompt( )  
console.log( )  
window.alert( )  
document.write( )
```

```
<!DOCTYPE html>
<html>
<head>
  <title>Instrucciones básicas de iniciación</title>
  <meta charset="utf-8">
  <meta name="author" content="Francesc Ricart">
</head>
<body>

<script type="text/javascript">

  // PRE . Programa que pregunta un dato al usuario y lo escribe por la consola del navegador
  y por el documento web.

  //1. Declarar variables
  var datoUsuario;

  //2. Funciones. En este caso no hay.

  //3. Calcular o instrucciones
  datoUsuario = window.prompt("Escribe un dato");

  //4. Devolver resultados
  document.write(datoUsuario);
  document.write("<br>");
  document.write("datoUsuario"); // mal

  console.log(datoUsuario);
  console.log("datoUsuario"); // mal

  // POST. Se ha escrito por pantalla el dato introducido por el usuario en una ventana
  emergente. También se ha escrito por la consola del navegador.
</script>

</body>
</html>
```



#### **Barcelona**

Francesc Tàrraga 14  
08027 Barcelona  
93 351 78 00

#### **Madrid**

Campanar 12  
28028 Madrid  
91 502 13 40

#### **Reus**

Alcalde Joan Bertran 34-38  
43202 Reus  
977 31 24 36

**[info@grupcief.com](mailto:info@grupcief.com)**

**[www.grupcief.com](http://www.grupcief.com)**

