

## UNIDAD 2 Empezando con JavaScript

### *Incorporar JS a una web*

**<script src="..."> </script>**

Un archivo JavaScript se puede vincular con la etiqueta

**<script src="..."></script>**, poniendo en el atributo **src** la ruta para llegar al archivo, de manera similar a como hacemos con una imagen. Para evitar ralentizar el renderizado de la página, se recomienda poner esta etiqueta al final de la página, antes del cierre de **</body>**. Alternativamente y para casos puntuales, podemos escribir el código JavaScript directamente dentro de la etiqueta

**<script> ... </script>** y no usar el atributo **src**.

#### **defer y async**

Estos dos atributos modifican la manera en la que el navegador carga o ejecuta JavaScript, y se ponen en la etiqueta de apertura del **<script>**. Si añadimos **defer**, el navegador espera a que el contenido del documento se haya cargado antes de iniciar el script. Si añadimos **async**, el navegador no bloqueará la carga de la página mientras está descargando el archivo.

#### **Orden de ejecución del script**

Por defecto (salvo que usemos atributos como **async** y **defer**) los archivos JS se ejecutan en orden de aparición, por lo que es importante que los archivos que puedan tener dependencias se carguen después que dichas dependencias. Dentro de un archivo, el código que escribamos también se interpreta de arriba abajo, aunque como veremos, tenemos mecanismos para crear repeticiones y retrasar la ejecución de ciertas partes del código.

### *La consola del navegador*

**console.log(...)**

La consola del navegador se puede encontrar en una de las pestañas del inspector, y nos permite ver si nuestro código tiene errores, pero también mostrar ciertos valores impresos. Si introducimos la instrucción **console.log()** dentro de nuestro código, escribirá en la consola el argumento que pasemos dentro del paréntesis, como puede ser una variable o un texto (los valores textuales se deben entrecomillar).

## Sintaxis JS

### Escritura

JavaScript diferencia entre mayúsculas y minúsculas, por lo que es importante ser consistente con la escritura de variables, nombres de funciones, etc, para que interprete nuestras instrucciones correctamente. Normalmente no se emplean guiones para los nombres de variables o propiedades (porque el `-` es el símbolo de operación de resta), por lo que en su lugar, las palabras compuestas se suelen escribir con la inicial de cada palabra en mayúscula, a partir de la segunda.

Ejemplo: `nombreVariable`.

### Comillas

JavaScript utiliza comillas fundamentalmente para acotar valores que no deben ser interpretados directamente (suelen ser valores textuales). Se puede emplear indistintamente comillas dobles `" "` o comillas simples `' '`, pero si queremos poner comillas dentro de comillas, tendremos que usar alternativamente de un tipo y otro.

Ejemplo: `"un ejemplo de 'texto con comillas'"`

### Los paréntesis

Las funciones en JavaScript reciben uno o varios argumentos (valores que interpretarán) a través de sus paréntesis. El nombre de la función que se ejecuta se escribe junto a los paréntesis, sin que haya un espacio en medio. Cuando pasamos múltiples argumentos en estos paréntesis, se escriben separándolos por comas.

Ejemplo: `console.log("Hola");`

### El punto

El punto en JavaScript indica pertenencia, normalmente a un objeto. Por ejemplo, en el código `console.log()`, estamos llamando a la función `log()`, que pertenece al objeto `console`.

### Punto y coma ; y llaves {}

De forma similar a CSS, las instrucciones finalizan normalmente con un `;` que indica que lo que viene a continuación es otra instrucción. También de manera similar a CSS, podemos agrupar un conjunto de instrucciones con las llaves `{ }` y emplearlas para vincularlas a otros elementos, como las funciones.

## Comentarios en JS

Los comentarios de bloque en JavaScript se escriben con `/* */`

Ejemplo: `/* texto dentro del comentario */`

Los comentarios de línea se escriben con `//` al principio de la línea a comentar.

Ejemplo: `// Línea comentada`

## Elementos JS

### Datos

Los datos son la información con la que trabajarán nuestros scripts. Los más habituales son los numerales (un valor numérico), literales (un valor textual, que se pone entre comillas para que JS no lo interprete directamente) y *booleanos* (cuyo valor es **true** o **false**, es decir, verdadero o falso).

### Variables

Las variables son términos clave cuyo valor puede cambiar. Nos permiten hacer referencia en nuestro código a datos que quizá desconozcamos en el momento, pero que serán interpretados y manipulados por nuestros *scripts*.

### Funciones

Las funciones permiten agrupar un conjunto de líneas de código para que sean ejecutadas solo cuando nos interese. Se declaran con la palabra **function** seguida del nombre de la función y paréntesis, y agrupan el código que escribamos a continuación entre llaves `{ }`. Para ejecutar una función previamente declarada, escribiremos su nombre junto a los paréntesis.

### Condicionales

Los condicionales nos permiten ejecutar un conjunto de líneas de código agrupadas solo en el caso de que se cumplan unas condiciones concretas (por ejemplo, que una variable sea igual a un valor concreto).

### Loops

Los *loops* nos permiten repetir la ejecución de un conjunto de líneas de código un número determinado de veces (por ejemplo, por cada elemento de un tipo que haya, o siempre y cuando se cumpla una condición).

## Objetos

Los objetos son modelos de datos que tienen propiedades (relación entre una clave y un valor asociado) y métodos (funciones internas). JavaScript entiende casi todo como objetos, como la ventana (el objeto **window**), la consola (el objeto **console**), e incluso las variables y las funciones, y como tales, estos objetos tienen propiedades y métodos a los que podemos acceder.

## Acceder al DOM

El DOM es el *Document Object Model*, es decir, el objeto documento (o cómo JavaScript interpreta una página web). Emplearemos métodos de este objeto, como **document.querySelector()**, para acceder al HTML y manipularlo.

## Eventos y listeners

Los eventos son sucesos en una página web, bien provocados por el usuario (hacer clic, hacer *scroll*, apretar una tecla...) o bien provocados por el propio navegador (iniciar la carga de una página, completarla, salir de una página...). Los *listeners* son métodos que nos permiten asociar una función concreta a un evento concreto, como puede ser que ante el hecho de que se haga clic en un botón, se ejecute una función de desplegado de un contenido.