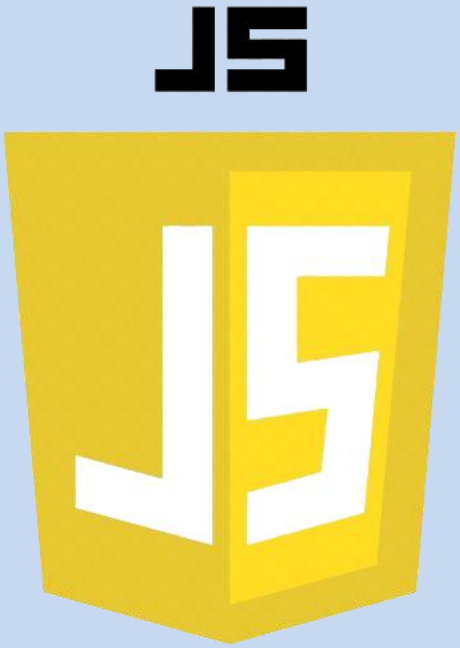


JavaScript

i

jQuery

Omar del Río García

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

CONTENIDO

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

INTRODUCCIÓN

TIPOLOGÍAS DE LENGUAJES DE PROGRAMACIÓN

Un **lenguaje de programación** es un idioma artificial diseñado para expresar procesos que pueden ser reproducidos por máquinas. Se utilizan para crear programas que controlan el comportamiento lógico de una máquina y para expresar algoritmos con precisión.

Decimos lenguaje porque está formado por un conjunto de símbolos, reglas sintácticas y semánticas que definen su estructura y significado de los elementos y expresiones.

Los **lenguajes interpretados** son aquellos que requieren un programa auxiliar o intérprete que traduce el lenguaje a binario para que la máquina lo pueda procesar y ejecutar. Ejemplos: PHP, Phyton, JavaScript, etc.

Los **lenguajes compilados** necesitan un programa anexo llamado compilador que hace la transformación a un lenguaje inteligible para la máquina y genera un archivo que se puede ejecutar sin la necesidad de ningún otro programa intermediario; es lo que se llama archivo ejecutable. Ejemplos: C, C++, Java, etc.

Los **lenguajes transpilados** son aquellos que, antes de ejecutarse, se transforman en otro lenguaje de nivel similar, habitualmente por motivos de compatibilidad o por aprovechar características avanzadas no disponibles en todas las plataformas. En este proceso, un transpilador convierte el código fuente original en otro código fuente equivalente pero más ampliamente soportado. Ejemplos: Haxe, Sass/SCSS, TypeScript, etc.

CONCEPTUALIZACIÓN DE LAS BASES DEL PENSAMIENTO COMPUTACIONAL

El pensamiento computacional se basa en pensar de la misma manera que lo haría un científico informático cuando nos enfrentamos a un problema. En otras palabras, es un proceso que permite formular problemas de manera que sus soluciones pueden ser representadas como secuencias de instrucciones y algoritmos.

Este tipo de pensamiento lo podemos definir como un proceso de reconocimiento de aspectos relacionados con la informática en la que se aplican herramientas y técnicas para comprender, razonar y solucionar problemas tanto naturales como artificiales. Estas características son la **abstracción**, el **pensamiento algorítmico**, la **descomposición** y el **reconocimiento de patrones**.

Es probable que durante el proceso de resolución de estos problemas exista información irrelevante. La **abstracción** es la característica de prescindir de la información irrelevante para que en la mesa esté sólo la información necesaria para el cumplimiento del objetivo.

El **pensamiento algorítmico** es otra de las características del pensamiento computacional: es necesario para comunicar e interpretar una serie de instrucciones ordenadas que nos lleven a un resultado concreto y predecible.

Es decir, el pensamiento algorítmico nos permite automatizar soluciones. Un buen ejemplo de este pensamiento es la cocina: las recetas son algoritmos en sí mismo. ¿Cómo se prepara un sándwich de mantequilla de cacahuete y mermelada? Pensar y escribir los pasos necesarios, sin olvidar ningún detalle, estructura el pensamiento de una forma computacional.

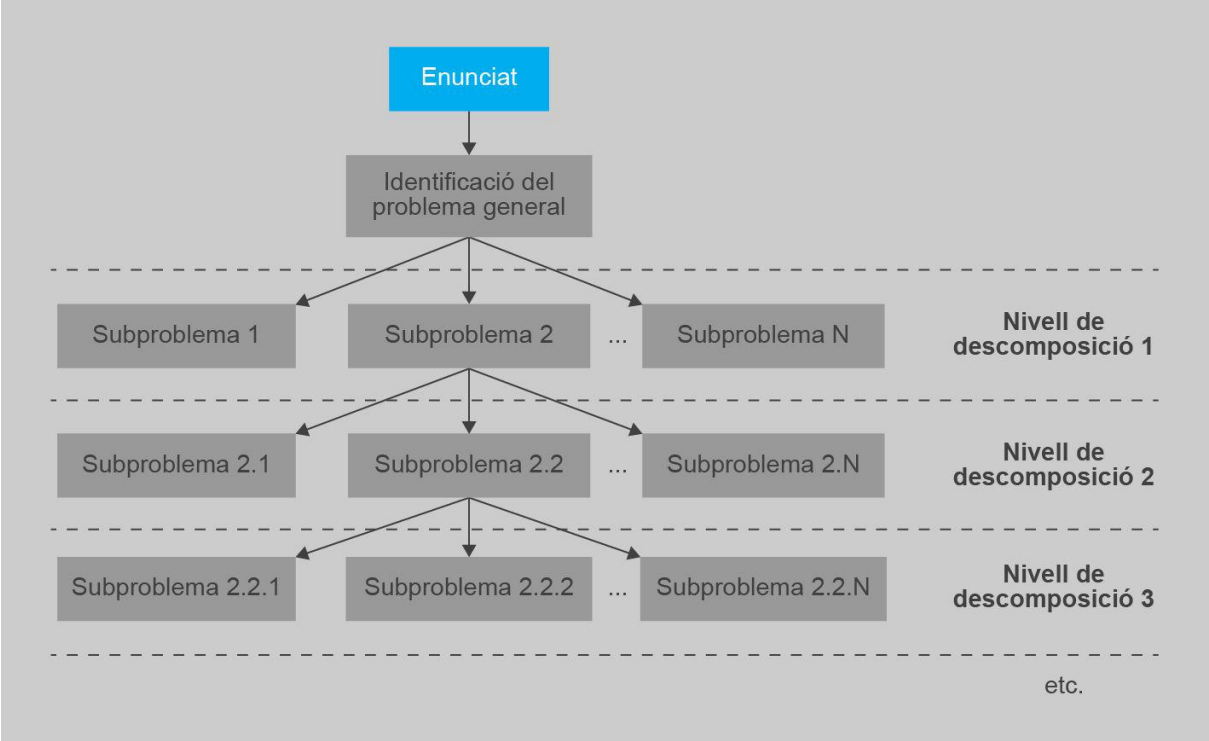
Generalitat de Catalunya
MINISTERIO DE EDUCACIÓN, FORMACIÓN PROFESIONAL Y DEPORTES
MINISTERIO DE TRABAJO Y ECONOMÍA SOCIAL
SERVICIO PÚBLICO DE EMPLEO ESTATAL SEPE SERVEI PÚBLIC D'OCUPACIÓ ESTATAL

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

Otra característica del pensamiento computacional es la **descomposición**: al enfrentarse a un problema, éste debe desarticularse para convertirlo en una práctica más sencilla.

Si el problema original es demasiado complejo para solucionarlo de golpe, hay que descomponerlo en diferentes problemas menores cada vez más específicos y concretos hasta que sean solucionables.

Esta forma de plantear subproblemas cada vez más concretos a partir de un problema general se llama diseño descendente.



Una vez desarticulado el problema principal, cada uno de ellos debe resolverse sobre la base de una metodología similar que se haya utilizado con otros problemas ya resueltos.

Esta característica es el **reconocimiento de patrones**: saber generalizar un proceso de resolución con la finalidad de que éste sirva para poder resolver otros problemas similares.

Cuando pensamos en los problemas, podemos reconocer similitudes entre ellos y que se pueden resolver de manera similar. A esto se denomina coincidencia de patrones, y es algo que hacemos naturalmente todo el tiempo en nuestra vida diaria.

Pensamiento computacional y programación no son sinónimos, pero comparten procesos similares: ambos son un medio que sirve para descomponer y resolver problemas. Mientras que el pensamiento computacional es aplicable a muchas disciplinas, la programación limita estos procesos exclusivamente en el ámbito de la informática.

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

JAVASCRIPT

JavaScript es un lenguaje de programación utilizado para crear pequeños programas encargados de realizar acciones dentro del ámbito de una página web. Con JavaScript podemos crear efectos especiales en las páginas y definir interactividades con el usuario. El navegador del cliente es el encargado de interpretar las instrucciones JavaScript y ejecutarlas para realizar estos efectos e interactividades, de manera que el mayor recurso, y quizás el único, con el que cuenta este lenguaje es el propio navegador.

JavaScript es un lenguaje interpretado que se introduce en una página web HTML. Un lenguaje interpretado quiere decir que a las instrucciones las analiza y procesa el navegador en el momento que deben ser ejecutadas.

Entre las acciones típicas que se pueden realizar en JavaScript tenemos dos vertientes. Por un lado los efectos especiales sobre páginas web, para crear contenidos dinámicos y elementos de la página que tengan movimiento, cambian de color o cualquier otro dinamismo. Por otro, JavaScript nos permite ejecutar instrucciones como respuesta a las acciones del usuario, de manera que podemos crear páginas interactivas con programas como calculadoras, agendas, o tablas de cálculo.

JavaScript es un lenguaje con muchas posibilidades, permite la programación de pequeños scripts, pero también de programas más grandes, orientados a objetos, con funciones, estructuras de datos complejos, etc. Toda esta potencia de JavaScript se pone a disposición del programador, que se convierte en el verdadero propietario y controlador de cada cosa que pasa en la página. Todo lo que veremos a continuación nos servirá de base para adentrarnos más adelante en el desarrollo de páginas enriquecidas del lado del cliente.

JavaScript, al igual que ActionScript en Flash o Visual Basic Script, es una de las múltiples maneras que han surgido para extender las capacidades del lenguaje HTML (lenguaje para el diseño de páginas de Internet). Al ser la más sencilla, es de momento la más extendida. JavaScript no es un lenguaje de programación propiamente dicho como C, C++, Delphi, etc. Es un lenguaje script u orientado a documento, como pueden ser los lenguajes de macros que tienen muchos procesadores de texto y hojas de cálculo. No se puede desarrollar un programa con JavaScript que se ejecute fuera de un navegador, aunque en este momento empieza a expandirse a otras áreas como la programación en el servidor con **NODE.JS**.

UN POCO DE HISTORIA SOBRE JAVASCRIPT

Según va creciendo la web y sus diferentes usos se fueron complicando las páginas y las acciones que se querían realizar a través de ellas. Al poco tiempo quedó reflejado que HTML no era suficiente para realizar todas las acciones que se pueden llegar a necesitar en una página web. En otras palabras, HTML se había quedado corto ya que sólo servía para presentar el texto en una página, definir su estilo y poco más.

Al complicar los sitios web, una de las primeras necesidades fue que las páginas respondieran a algunas acciones del usuario, para desarrollar pequeñas funcionalidades más allá de los propios enlaces. El primer ayudante para cubrir las necesidades que estaban surgiendo fue Java, que es un lenguaje de propósito general, pero que había creado una manera de incrustar programas en páginas web. A través de la tecnología del Applets, se podía crear pequeños programas que se ejecutaban en el navegador dentro de las propias páginas web, pero que tenían posibilidades similares a los programas de propósito general. La programación de Applets fue un gran avance y Netscape, entonces el navegador más popular, había roto la primera barrera del HTML al hacer posible la programación dentro de las páginas web. No hay duda de que la aparición de los Applets supuso un gran avance

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

en la historia de la web, pero no ha sido una tecnología definitiva y muchas otras han seguido implementando el camino que comenzó con ellos.

Netscape, después de hacer sus navegadores compatibles con los applets, comenzó a desarrollar un lenguaje de programación al que llamó LiveScript que permitió crear pequeños programas en las páginas y que fuera mucho más sencillo de utilizar que Java. De manera que el primer JavaScript se llama LiveScript, pero no duró mucho este nombre, ya que antes de lanzar la primera versión del producto se forjó una alianza con Sun Microsystems, creador de Java, para desarrollar en conjunto este nuevo lenguaje.

La alianza hizo que el JavaScript se diseñara como un hermano pequeño de Java, sólo útil dentro de las páginas web y mucho más fácil de utilizar, de manera que cualquier persona, sin conocimientos de programación, pudiera adentrarse en el lenguaje y utilizarlo a su aire. Además, para programar JavaScript solo es necesario un kit de desarrollo, ni compilar los scripts, ni realizarlos en ficheros externos al código HTML, como pasaba con los applets.

Netscape 2.0 fue el primer navegador que entendía JavaScript y su iniciativa fue seguida por otros clientes web como Internet Explorer a partir de la versión 3.0. Sin embargo, la compañía Microsoft nombró a este lenguaje como JScript y tenía ligeras diferencias respecto a JavaScript, algunas de las cuales perduran hasta el día de hoy.

Diferencias entre diferentes navegadores

Como hemos dicho el JavaScript de Netscape y el de Microsoft Internet Explorer tenía ligeras diferencias, pero es que también el mismo lenguaje evolucionó a medida que los navegadores presentaban sus diferentes versiones a medida que las páginas web se hacían más dinámicas y más exigentes las necesidades de funcionalidades.

Las diferencias de funcionamiento de JavaScript ha marcado la historia del lenguaje y la manera en que los desarrolladores se relacionan con él, debido a que estaban obligados a crear código que funcionara correctamente en diferentes plataformas y diferentes versiones de las mismas. Hoy en día, siguen habiendo muchas diferencias y para solucionarlo han surgido muchos productos como los Frameworks JavaScript, que ayudan a realizar funcionalidades avanzadas de DHTML sin tener que preocupar en hacer versiones diferentes de los scripts, para cada uno de los navegadores posibles de mercado.

Diferencias entre Java y JavaScript

Realmente JavaScript se llamó así porque Netscape, que estaba aliado a los creadores de Java en la época, quiso aprovechar el conocimiento y la percepción que las personas tenían del popular lenguaje. Con todo, se creó un producto que tenía ciertas similitudes, como la sintaxis del lenguaje o el nombre. Se hizo entender que era un hermano pequeño y orientado específicamente para hacer cosas en las páginas web, pero también se hizo caer a muchas personas en el error de pensar que son lo mismo. Queremos que quede claro que el JavaScript no tiene nada que ver con Java, excepto en sus orígenes, como se ha podido leer hace unas líneas. Actualmente, son productos totalmente diferentes y no guardan entre sí más relación que la sintaxis idéntica y poco más. Algunas diferencias entre estos dos lenguajes son las siguientes: el Compilador. Para programar en Java necesitamos un Kit de desarrollo y un compilador. Sin embargo, JavaScript no es un lenguaje que necesite que sus programas se compilen, sino que estos se interpretan por parte del navegador cuando éste lee la página.

- **Orientado a objetos**: Java es un lenguaje de programación orientado a objetos. (Más tarde veremos que quiere decir orientado a objetos, para quien no lo sepa todavía). JavaScript se ha actualizado para

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

que también sea orientado a objetos, pero podemos programar sin necesidad de crear clases, tal y como se realiza en los lenguajes de programación estructurada como C o Pascal.

- **Propósito**: Java es mucho más potente que JavaScript, esto se debe a que Java es un lenguaje de propósito general, con el que se pueden hacer aplicaciones de lo más variado, sin embargo, con JavaScript solo podemos escribir programas para que se ejecuten en páginas web.
- **Estructuras fuertes**: Java es un lenguaje de programación fuertemente tipado, esto quiere decir que al declarar una variable deberemos indicar su tipo y no podrá cambiar de un tipo a otro automáticamente. Por su parte JavaScript no tiene esta característica, y podemos colocar en una variable la información que deseamos, sin importar el tipo de la misma. Además, podremos cambiar el tipo de información de una variable cuando queramos.
- **Otras características**: Java es mucho más complejo, aunque también más potente, robusto y seguro. Tiene más funcionalidades que JavaScript y las diferencias que los separan son lo suficientemente importantes como para distinguirlos fácilmente.

Qué necesitas para trabajar con JavaScript

Para programar en JavaScript necesitamos básicamente lo mismo que para desarrollar páginas web con HTML: un **entorno integrado de desarrollo** o **IDE** (acrónimo en inglés de *integrated development environment*) o editor de texto y un navegador compatible con JavaScript.

Diferentes versiones de JavaScript, los navegadores que las aceptan y sus avances.

El lenguaje ha ido avanzando durante sus años de vida e incrementando sus capacidades. Al principio podía hacer muchas cosas en la página web, pero tenía pocas instrucciones para crear efectos especiales.

Con el tiempo también el HTML ha avanzado y se han creado nuevas características como las capas, que permiten tratar y maquetar los documentos de manera diferente. JavaScript ha avanzado también y para gestionar todas estas nuevas características se han creado nuevas instrucciones y recursos.

Realmente cualquier navegador medianamente moderno tendrá ahora todas las funcionalidades de JavaScript que necesitaremos. No obstante, puede ir bien conocer las primeras versiones de JavaScript que comentamos, a modo de curiosidad.

- **JavaScript1**: nació con el Netscape 2.0 y soportaba gran cantidad de instrucciones y funciones, casi todas las que existen ahora ya se introdujeron en el primer estándar.
- **JavaScript1.1**: es la versión de JavaScript que se diseñó con la llegada de los navegadores 3.0. Implementaba poco más que su anterior versión, como el tratamiento de imágenes dinámicamente y la creación de arrays.
- **JavaScript1.2**: la versión de los navegadores 4.0. Esta tiene como desventaja que es un poco diferente en plataformas Microsoft y Netscape, ya que ambos navegadores crecieron de manera diferente y estaban en plena lucha por el mercado.
- **JavaScript1.3**: versión que implementan los navegadores 5.0. En esta versión se han limado algunas diferencias entre los dos navegadores.
- **JavaScript 1.5**: Versión que implementa Netscape 6.
- Por su parte, Microsoft también ha evolucionado hasta presentar su versión 5.5 de JScript (así llamamos al JavaScript utilizado por los navegadores de Microsoft).

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

- **ECMAScript**: En 1997 los autores propusieron Javascript como estándar de la European Computer Manufacturers Association ECMA, que a pesar de su nombre no es europeo, sino internacional, con la sede en Ginebra.
- Para evitar estas incompatibilidades, el World Wide Web Consortium (W3C) diseñó el estándar Document Object Model (DOM, o Modelo de Objetos del Documento en castellano), que incorpora el Konqueror, las versiones 6 de Internet Explorer y Netscape Navigator, Opera versión 7, y Mozilla desde su primera versión.

SINTAXIS BÁSICA

JavaScript es un lenguaje de programación y, tal y como hemos comentado antes, está formado por un conjunto de símbolos, reglas sintácticas y semánticas que definen su estructura y significado de los elementos y expresiones.

Para hacernos una idea, cuando redactamos un contenido, la frase u oración es el conjunto de palabras con sentido completo, y normalmente las acabamos con un signo de puntuación. A esta estructura le decimos sentencia en programación.

A su vez, las oraciones se componen de unidades gramaticales o sintagmas: sujeto, predicado, complemento directo, complemento indirecto,... Dentro de una sentencia, a estas unidades le diremos expresiones.

En las frases, podemos tener sustantivos para hacer referencia a algún concepto. En la programación, tenemos las variables que almacenan información.

En las oraciones tenemos verbos para expresar acciones. En la programación, tenemos comandos para dar las órdenes.

En las frases tenemos conjunciones para poder unir palabras y sintagmas. En la programación tenemos los operadores.

ORACIONES.....SENTENCIES
SINTAGMAS ..... EXPRESIONES
SUSTANTIVOS .....VARIABLES
CONJUNCIONES ..... OPERADORES
VERBOS ..... COMANDOS

Cada vez que escribimos una instrucción hay que acabar con el carácter punto y coma. Es importantísimo tener en cuenta que JavaScript es sensible a mayúsculas y minúsculas.

Aunque nos dejemos el punto y coma ";" al final de una sentencia, JavaScript si detecta un salto de línea al código la finaliza implícitamente, aunque es una buena práctica finalizarlas explícitamente para mejorar el rendimiento del navegador.

COMENTARIOS

Una parte fundamental de la programación es añadir comentarios que nos ayuden a componer cada parte del código sin que el navegador los interprete. Para añadir comentarios tenemos la opción de añadir un comentario

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

de una sola línea **//** y cuando cambiamos de línea continuamos con la ejecución normal, o la opción de abrir un comentario **/*** y no cerrarlo ***/** hasta más adelante:

```
// comentario de una línea

/*
      Comentario
      en varias
      líneas de código
*/
```

FORMAS DE EJECUTAR SCRIPTS DE JAVASCRIPT

Hay dos maneras básicas de ejecutar scripts JavaScript en una página: al cargar la página o como respuesta a acciones del usuario.

EJECUCIÓN DIRECTA

Es el método de ejecutar scripts más básico. En este caso se incluyen las instrucciones dentro de la etiqueta **<script>**. Cuando el navegador lee la página y encuentra un script interpreta las líneas de código y las va ejecutando una después de otra. Llamamos a este modo ejecución directa ya que cuando se lee la página se ejecutan directamente los scripts.

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Ejemplo JS</title>
    <style>
        body {font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;}
    </style>
</head>
<body>
    <h1>Ejemplo JS</h1>
    <script>
        document.write("¡Hola mundo!");
    </script>
</body>
</html>
```

En un mismo documento .html podemos tener tantas etiquetas `<script>` como hagan falta en cualquier parte del documento, aunque lo más habitual es que estén en el `<head>` del documento.

RESPUESTA A UN EVENTO

Es la otra manera de ejecutar scripts, pero antes de verla tenemos que definir los eventos: son las acciones que realiza el usuario. Los programas como JavaScript están preparados para atrapar determinadas acciones realizadas, en este caso sobre la página, y realizar acciones como respuesta. De esta manera se pueden realizar programas interactivos, ya que controlamos los movimientos del usuario y respondemos a ellos. Hay muchos tipos de eventos diferentes, por ejemplo la pulsación de un botón, el movimiento del ratón o la selección de texto de la página.

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

Las acciones que queremos hacer como respuesta a un evento deben indicarse dentro del mismo código HTML, pero en este caso se indican en atributos HTML que se colocan dentro de la etiqueta que queremos que responda a las acciones del usuario.

```
<button type="button" onclick="alert("¡Hola mundo!");">Saluda</button>
```

INCLUIR FICHEROS EXTERNOS DE JAVASCRIPT

Otra manera de incluir scripts en páginas web, implementada a partir de JavaScript 1.1, es incluir archivos externos donde se pueden colocar muchas funciones que se utilicen en la página. Los ficheros suelen tener extensión **.js** y se incluyen de esta manera:

```
<script src="archivo_externo.js" defer>
// estoy incluyendo el fichero " archivo_externo.js "
</script>
```

Esta es la forma más habitual y adecuada para mantener más organizado el código en nuestros proyectos web.

En caso de vincular con un archivo .js, aparte del atributo src podemos añadir otros atributos para definir cuándo se ejecutará este código:

- **src**: especifica la URL de un fichero de script externo.
- **async**: especifica que el script se descarga en paralelo a la página, y se ejecuta tan pronto como está disponible (antes de que acabe la carga de la página).
- **defer**: especifica que el script se descarga en paralelo a la página, y se ejecuta después de que la página haya terminado de cargarse.

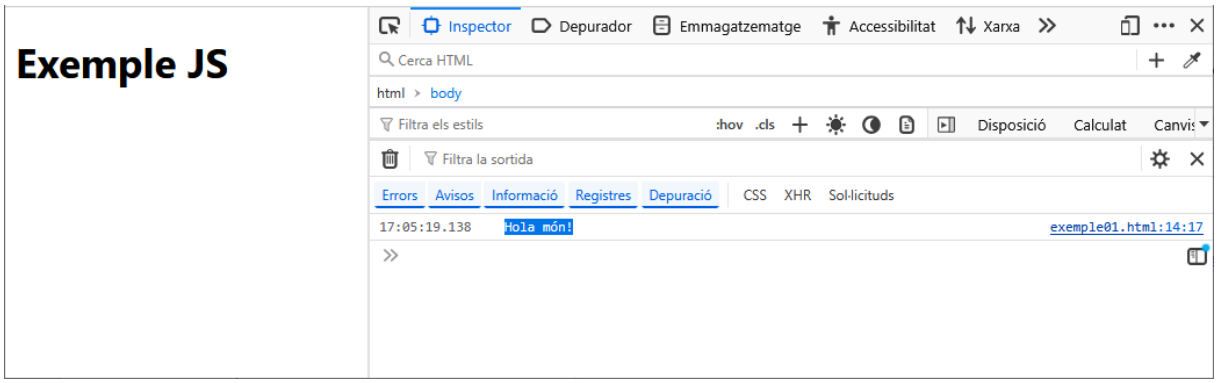
Es importante aclarar que si utilizamos una etiqueta `<script>` para enlazar a otro archivo .js, no podemos utilizar esta misma etiqueta para añadir código, porque será sobreescrito por el documento enlazado por el atributo src.

DEPURACIÓN DEL CÓDIGO

Una forma habitual y rápida para depurar o mostrar resultados del código es emplear el objeto **console** con los diferentes métodos, el más utilizado el **.log()**. Con la expresión **console.log("***mensaje***")** podemos lanzar mensajes en la consola de depuración del propio navegador:

```
<script>
        console.log("Hola món!");
</script>
```

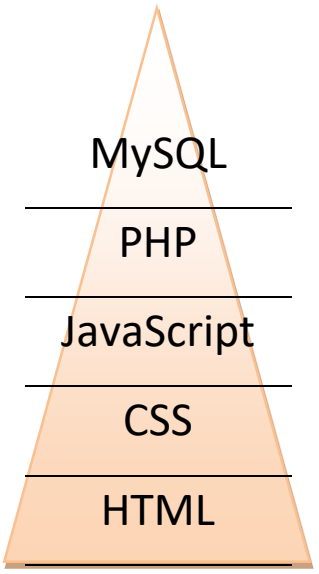

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)



El objetivo de este manual no es mostrar todas las opciones posibles porque es un lenguaje vivo que cambia continuamente, por lo tanto, recomiendo consultar blogs, foros y otras web de referencia como por ejemplo http://www.w3schools.com/ o https://developer.mozilla.org/

De forma rápida, podemos listar los lenguajes web como una pirámide, donde cada uno de ellos complementa el anterior:

- En la base tenemos el **HTML**, que nos permite estructurar los contenidos por su representación.
- El **CSS** complementa al HTML para darle estilo y diseño, una mejor apariencia a la estructura anterior.
- El **JavaScript** da dinamismo y efectos especiales al HTML y al CSS. La suma de los tres lenguajes se denomina DHTML, y su principal característica es que sólo hace falta un navegador para ver los resultados.
- El **PHP** es un lenguaje de servidor, por lo tanto, permite aumentar la seguridad y utilizar recursos de forma independiente en el navegador del usuario.
- El **MySQL** es una tecnología de base de datos, que le permite al PHP almacenar datos y contenidos para recuperarlos posteriormente; es la base para desarrollar herramientas web como los gestores de contenidos.



Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

VARIABLES

Uno de los fundamentos de la programación es conocer el estado del sistema en todo momento y lo conseguimos acumulando información en las variables. Podemos imaginarnos las variables como cajones dentro de la memoria RAM del dispositivo, unos cajones que creamos según los necesitamos pero que se eliminarán automáticamente cuando cambiemos de documento. Podemos crear tantos cajones -variables- como necesitemos, pero para poder organizarnos y poder encontrar esta información hay que etiquetar estos cajones, es decir, hay que nombrar las variables.

Las variables son contenedores de información para poder recuperarla o actualizarla en cualquier momento a partir de su nombre.

En el momento de crear una variable hay que darle un nombre. Este nombre es completamente arbitrario, pero hay que seguir unas normas básicas para asegurarnos el buen funcionamiento:

- No podemos utilizar palabras ya reservadas para el propio lenguaje JavaScript: las nombradas en este manual.
- No podemos emplear espacios ni signos de puntuación: los nombres deben escribirse seguidos, sin acentos ni otros caracteres especiales, empleando el alfabeto inglés; sólo podemos utilizar el guión bajo "_" siguiendo la práctica *snake_case* o el símbolo de "$".
- Pueden contener números, pero nunca como primer carácter: el primer carácter siempre será una letra.
- Como buena práctica, se recomienda que tengan nombres autodefinitorios: esto implica que los nombres sean compuestos por diferentes palabras.
- Como buena práctica, se recomienda utilizar la práctica del *lowerCamelCase*: escribir frases o palabras compuestas eliminando los espacios y poniendo en mayúscula la primera letra de cada palabra, excepto la primera palabra que se mantiene en minúscula.

Tipos de información	Nombre de variable incorrecto	Nombre de variable correcto
Nombre del cliente	N-C	nomClient
1º número	1 número	numero1
Número 2	Número2	num_2

DECLARACIÓN E INSTANCIACIÓN

Al proceso de crear una variable asignándole un nombre único e irrepetible se le dice **declaración**. Es importante tener en cuenta que JavaScript permite sobreescribir las variables si al declarar una variable le asignamos un nombre ya existente, provocando errores posteriores en el tratamiento en la información.

La **instanciación** es el proceso de asignar un valor a la variable, y se hace con el operador "=". Si intentamos hacer una instanciación sin declarar previamente la variable, JavaScript hará una declaración implícita y nos dejará continuar sin errores, pero es una mala práctica porque se genera código muy confuso.

Para declarar una variable utilizamos uno de los comandos reservados para el propio lenguaje:

Tipo de comando	Ejemplo de declaración e instanciación	Cuando lo utilizaremos
var	var nomClient = “Maria”;	Es el método clásico y lo utilizaremos cuando queremos una variable de ámbito global.

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

let	let contador = 1;	Es el método más moderno y lo utilizaremos cuando queremos una variable de ámbito local.
const	const nomClient = "Jordi";	Es el método más moderno para definir constantes: variables que una vez instanciadas no se volverán a cambiar.

Cuando utilizamos **var** o **let** podemos hacer la declaración en una sentencia y la instanciación en otra sentencia, pero cuando empleamos **const** hay que hacer la declaración y la instanciación en la misma sentencia.

Podemos declarar múltiples variables simultáneamente con un único comando **var** o **let** si separamos los diferentes nombres con comas ",".

```
<script>
    // Declaración de variables
    var nombreCliente, apellidoCliente;

    // Instanciación de variables
    nombreCliente = "Martín";
    apellidoCliente = "García";

    // Declaración e instanciación de constante
    const idioma = "es";
</script>
```

ÁMBITO DE VARIABLES –SCOPE-

En la definición de **var** y **let** hemos comentado que la principal diferencia es el ámbito de su uso –scope-: se le llama ámbito de las variables en el lugar donde éstas están disponibles. En general, cuando declaramos una variable hacemos que esté disponible en el lugar donde se ha declarado. Esto ocurre en todos los lenguajes de programación y, como JavaScript se define dentro de una página web, las variables que declaramos en la página estarán accesibles dentro de ella.

En JavaScript no podremos acceder a variables que hayan sido definidas en otra página. Por lo tanto, la propia página donde se define es el ámbito más habitual de una variable y la llamaremos a este tipo de variables globales en la página. Veremos también que se pueden hacer variables con ámbitos diferentes del global, es decir, variables que declararemos y tendrán validez en lugares más acotados.

VARIABLES GLOBALES

Como hemos dicho, las variables globales son las que están declaradas en el ámbito más amplio posible, que en JavaScript es una página web. Para declarar una variable global en la página simplemente lo haremos con la palabra **var**.

```
<script>
    var variableGlobal;
</script>
```

Las variables globales son accesibles desde cualquier lugar de la página, es decir, desde el script donde se han declarado y todos los demás scripts de la página, incluidos los manejadores de eventos, como el onclick, que ya vimos que se podía incluir dentro de determinadas etiquetas HTML.

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

VARIABLES LOCALES

También podremos declarar variables en lugares más acotados, como por ejemplo una función: a estas variables las llamaremos locales. Cuando se declaren variables locales sólo podremos acceder dentro del lugar donde se ha declarado, es decir, si la habíamos declarado en una función sólo podremos acceder cuando estemos en esta función.

Las variables pueden ser locales a una función, pero también pueden ser locales en otros ámbitos, como por ejemplo un bucle. En general, son ámbitos locales cualquier lugar acotado por llaves. Estos elementos se tratan más adelante en este manual.

TIPOS DE VARIABLES

JavaScript no es un lenguaje fuertemente tipado, y esto quiere decir que las variables no tienen un tipo definido en su declaración, sino que se definen según el tipo de valor que le asignamos en la instanciación. Así, según el valor que instanciemos a una variable, tendremos los siguientes tipos:

Tipo de variable	Ejemplo	Explicación
string	`var nombreCliente = "María";`	Cadena de texto. Los valores van entre comillas dobles “ " “ o sencillas “ ' “.
number	`var registro = 1.5;`	Números. Los valores sólo pueden ser numéricos; el separador decimal es el punto ".".
boolean	`var estado = true;`	Los valores boleanos son los valores reservados **true** o **false**.
undefined	`var edad;`	Es el valor predeterminado de las variables declaradas pero ni instanciadas.
null	`var apellidoCliente = null;`	Es un tipo de valor que es “sin valor”; se puede utilizar para dejar una variable vacía o vaciar de valor una variable ya instanciada previamente.
function	`var miFuncion = function(){};`	Una de las formas de declarar funciones es como una variable. Las funciones se explican más adelante.
object	`var persona = {}`	Los objetos son la forma de almacenar información de forma más estructurada y compleja. Los objetos se explican más adelante.

Definir correctamente el tipo de las variables nos permite utilizar los operadores de la forma que más nos interese, por lo tanto hay que controlar el tipo de las variables.

Hay que insistir en que el tipo de una variable se establece en la instanciación, por lo tanto, si una variable se vuelve a instanciar con un tipo de valor diferente, la variable cambia también de tipo. Para saber en cada momento el tipo de una variable podemos emplear el operador **typeof**:

```
<script>
        var laMevaVariable;
        console.log(laMevaVariable, typeof laMevaVariable); // undefined undefined

        laMevaVariable = "Jordi";
        console.log(laMevaVariable, typeof laMevaVariable); // Jordi string

        laMevaVariable = 2026;
        console.log(laMevaVariable, typeof laMevaVariable); // 2026 number
```


Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

```
        laMevaVariable = true;
        console.log(laMevaVariable, typeof laMevaVariable); // true boolean

        laMevaVariable = null;
        console.log(laMevaVariable, typeof laMevaVariable); // null object

        laMevaVariable = {};
        console.log(laMevaVariable, typeof laMevaVariable); // Object {} object
    </script>
```

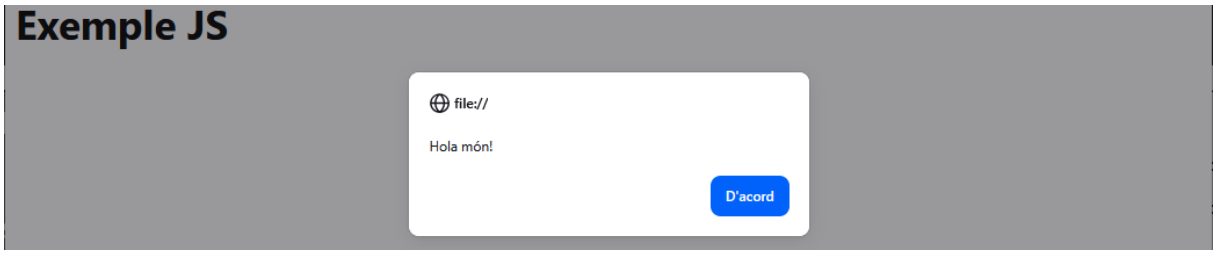
COMANDOS DE SALIDA Y ENTRADA DE VALORES

Una vez tenemos valores almacenados en las variables, sólo hay que nombrar su nombre para hacer referencia a su contenido. Pero si queremos ver su valor en pantalla, hay que utilizar alguno de los comandos del lenguaje. Ya hemos visto `console.log()`, pero tenemos otros para poder mostrar directamente en pantalla:

ALERT

Con el comando **`alert()`** podemos mostrar mensajes cortos en una pequeña ventana del propio navegador; sólo hay que poner el mensaje tipo texto o el nombre de la variable entre los paréntesis:

```
<script>
    const mensaje = "Hola món!";
    alert(mensaje);
</script>
```

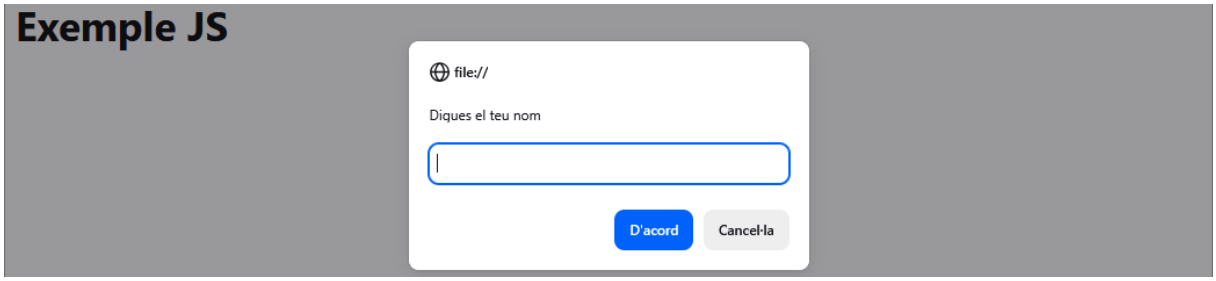


PROMPT

Con el comando **`prompt()`** podemos pedir información en una pequeña ventana del propio navegador para que el visitante llene con su información; sólo hay que poner nuestra pregunta tipo texto y utilizar este prompt para instanciar una variable:

```
<script>
    const nomVisitant = prompt("Digues el teu nom");
    alert(nomVisitant);
</script>
```

El valor que devuelve el prompt es siempre un tipo string.



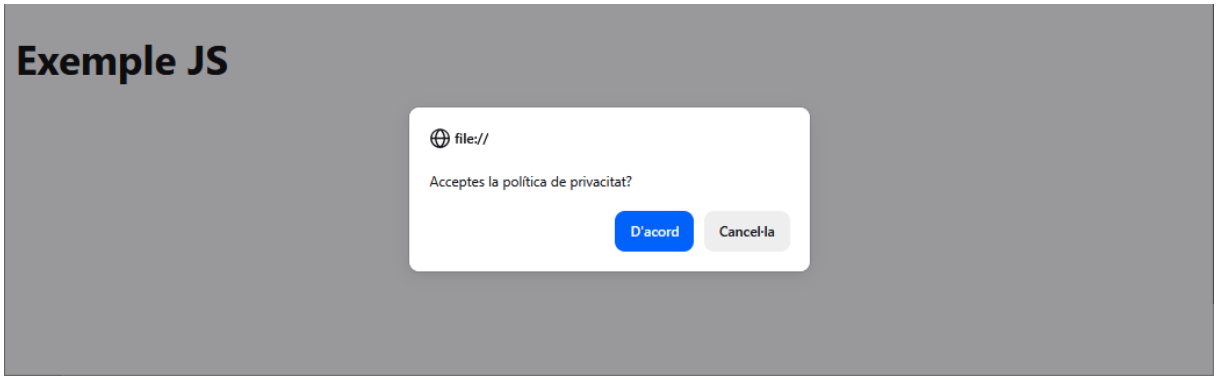
Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

CONFIRM

Con el comando **`confirm()`** podemos mover información en una pequeña ventana del propio navegador para que el visitante la acepte (si pulsa el botón *Aceptar*) o la rechace (si pulsa el botón *Cancelar*); solo hay que poner nuestra pregunta tipo texto y utilizar este confirm para instanciar una variable:

```
<script>
    const aceptacion = confirm("Acceptes la política de privacitat?");
    alert(aceptacion);
</script>
```

El valor que devuelve el confirm es siempre un tipo Boolean (true o false).



CONVERSIÓN DEL TIPO DE VALORES

Si queremos convertir un valor tipo a otro, podemos emplear funciones generales del JavaScript que necesitan que le pongamos el valor original entre paréntesis y nos devuelve el mismo valor pero como un tipo diferente:

Función	Ejemplo	Explicación

Boolean()	`var x = "1"; // string`
	`x = Boolean(x);`
	`// Boolean true`
		Devuelve un booleano convertido desde cualquier otro tipo de valor.

Number()	`var x = "4.5"; // string`
	`x = Number(x);`
	`// number 4.5`
		Devuelve un número convertido desde una cadena de texto.

parseFloat()	`var x = "4.5"; // string`
	`x = parseFloat(x);`
	`// number 4.5`
		Devuelve un número de punto flotante convertido desde una cadena de texto.

parseInt()	`var x = "4.5"; // string`
	`x = parseInt(x);`
	`// number 4`
		Devuelve un número entero convertido desde una cadena de texto.

String()	`var x = 6; // number`
	`x = String(x):`
	`// string '6'`
		Convierte el valor de un objeto en una cadena de texto.

```
<script>
    var edad = prompt("Dime tu edad"); // string
    edad = Number(edad); //number
</script>
```


Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

OPERADORES

Los operadores nos permiten aplicar cambios, cálculos o procesos sobre los valores de las variables según su tipo, por lo tanto es muy importante controlar el tipo de las variables para asegurarnos de que el operador se aplique correctamente. Así, categorizamos a los operadores según el tipo de variable a quien se aplica.

El primer operador que ya hemos visto es el de asignación para poder hacer las instanciaciones:

Tipo de operador	Ejemplo	Explicación
=	`nombreCliente = "María";`	Permite asignar un valor a una variable en una instanciación.

TEXTO

Los operadores de texto se pueden utilizar en las cadenas de texto y variables tipo string:

Tipo de operador	Ejemplo	Explicación
"	`nomClient = "Maria";`	Permite definir un valor tipo texto (string).
'	`cognomClient = 'Garcia';`	Permite definir un valor tipo texto (string).
+	`alert(nomClient + ' ' + cognomClient)`	Permite concatenar dos valores en una misma cadena de texto.
+=	`nomClient += ' ';` `nomClient += cognomClient;`	Asignación con concatenación: permite concatenar al mismo valor que ya está en una variable
`	`alert(`Benvingut/da ${nomClient} ${cognomClient}`);`	El acento abierto permite hacer *templates*: estructuras tipo texto donde podemos emplear libremente las otras comillas y donde añadimos valores JavaScript con **${}**
\"	`alert("Benvinguts a \"La Meva Web\"!");`	Comillas literales: con la barra de escape podemos añadir " sin que se interpreten como un operador.
\'	`alert(' Benvinguts a L\'Hospitalet');`	Comillas literal: con la barra de escape podemos añadir ' sin que se interprete como un operador.
\n	`alert("Benvinguts a:\n \"La Meva Web\"!");`	Salto de línea: con la barra de escape podemos añadir un salto de línea en una ventana alert, prompt o confirm.
\t	`alert("Benvinguts a:\n\t \"La Meva Web\"!");`	Tabulación: con la barra de escape podemos añadir una tabulación en una ventana alert, prompt o confirm.

NÚMEROS

Los operadores numéricos permiten hacer cálculos sobre valores tipo number:

Tipo de operador	Ejemplo	Explicación
+	`var num = 7 + 3; // 10`	Suma.
-	`var num = 7 - 3; // 4`	Resta.
*	`var num = 7 * 3; // 21`	Producto.
/	`var num = 7 / 3; // 2,33`	Fracción.
%	`var num = 7 % 3; // 1`	Módulo: devuelve el residuo de una división.
+=	`var num = 7;` `num += 3; // 10`	Asignación con suma: suma el número al valor de la variable.

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

-=	`var num = 7;` `num -= 3; // 4`	Asignación con resta: resta el número al valor de la variable.
***=**	`var num = 7;` `num *= 3; // 21`	Asignación con multiplicación: multiplica el número al valor de la variable.
/=	`var num = 7;` `num /= 3; // 2.33`	Asignación con fracción: hace la fracción del número al valor de la variable.
++	`var num = 7;` `num ++; // 8`	Incremento unitario: suma 1 al valor de la variable.
--	`var num = 7;` `num --; // 6`	Decremento unitario: resta 1 al valor de la variable.

```
<script>
    var edad = prompt("Dime tu edad"); // string
    edad = parseInt(edad); //number
    edad ++;
    alert(`¡El próximo aniversario cumplitàs ${edad} años!`);
</script>
```

LÓGICOS Y DE COMPARACIÓN

Estos operadores de comparación devuelven siempre un valor boolean de *true* o *false*:

Tipo de operador	Ejemplo	Explicación
==	`var comp = 7 == 3; // false`	Igualdad: compara dos valores.
===	`var comp = 7 === '7'; //false`	Identidad: compara dos valores y su tipo.
!=	`var comp = 7 != 3; // true`	Diferencia.
>	`var comp = 7 > 3; // true`	Mayor que: compara dos valores.
>=	`var comp = 7 >= 3; // true`	Mayor o igual que: compara dos valores.
<	`var comp = 7 < 3; // false`	Menor que: compara dos valores.
<=	`var comp = 7 <= 3; // false`	Menor o igual que: compara dos valores.
&&	`var comp = 7 > 3 && 3 > 7;` `// false`	Operador Y lógico: concatena dos comparaciones.
||	`var comp = 7 > 3 || 3 > 7;` `// true`	Operador O lógico: concatena dos comparaciones.
!	`var comp = !true; // false`	Operador de negación: niega el valor.

La operación lógica **AND** obtiene su resultado combinando dos valores booleanos. El operador se indica mediante el símbolo && y su resultado solamente es *true* si los dos operandos son *true*:

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

La operación lógica **OR** también combina dos valores booleanos. El operador se indica mediante el símbolo || y su resultado es *true* si alguno de los dos operandos es *true*:

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

variable1	variable2	variable1 || variable2
true	true	**true**
true	false	**true**
false	true	**true**
false	false	**false**

En el siguiente ejemplo se plantea dos posibilidades: que el visitante acepte o no la política de privacidad y que la edad que ha informado sea superior o igual a 18. Si se cumplen ambas posibilidades (ambas tienen un valor *true*) tendremos *true* en la variable *controlAcceso*; si cualquiera de las variables *aceptacion* o *controlEdad* tienen un valor *false*, tendremos un *false* en la variable *controlAcceso*:

```
<script>
    var aceptacion = false, edad = 0, controlEdad = false, controlAcceso = false;
    aceptacion = confirm("Aceptas la política de privacidad?");
    edad = parseInt(prompt("Dime tu edad "));
    controlEdat = edad >= 18
    controlAcceso = aceptacion && controlEdat;
    alert(`¿Puedes acceder? ${controlAcceso}`);
</script>
```

PRIORIDAD DE LOS OPERADORES

Según el orden de prioridad, los operadores se ejecutan antes o después:

1. OPERADORES DE CÁLCULO DE 1º ORDEN:
    - + suma
    - - resto
    - * multiplicación
    - / división (fracción)
    - % residuo de una división
2. OPERADORES CONDICIONALES DE 2º ORDEN:
    - ==, ===
    - >=, <=
    - >, <
    - !=
3. OPERADORES LÓGICOS DE 3º ORDEN:
    - &&
    - ||
    - !
4. OPERADORES DE CÁLCULO DE 4º ORDEN:
    - +=; -=; *=; /=; %= operador matemático combinado
5. OPERADORES DE CÁLCULO DE 5º ORDEN:
    - ++ incremento unitario
    - -- decremento unitario

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

ESTRUCTURAS DE CONTROL

Los programas que se pueden realizar utilizando solamente variables y operadores son una simple sucesión lineal de instrucciones básicas: el navegador lee cada sentencia, la ejecuta una única vez y salta a la siguiente hasta que se acaba el documento.

No obstante, no se pueden realizar programas que muestren un mensaje si el valor de una variable es igual a un valor determinado y no muestren el mensaje en el resto de casos. Tampoco se puede repetir de manera eficiente una misma instrucción, como por ejemplo sumar un determinado valor a todos los elementos de un array.

Para realizar este tipo de programas son necesarias las **estructuras de control de flujo**, que son instrucciones del tipo "*si se cumple esta condición, hazlo; si no se cumple, haz eso otro*". También existen instrucciones del tipo "*repite esto mientras se cumpla esta condición*".

Las estructuras de control de flujo permiten controlar el flujo de ejecución del código delimitando qué parte del código se ejecuta o cuántas veces lo hace.

Si se utilizan estructuras de control de flujo, los programas dejan de ser una sucesión lineal de instrucciones para convertirse en programas inteligentes que pueden tomar decisiones en función del valor de las variables.

Las estructuras de control de flujo se caracterizan por cerrar el código a evaluar entre llaves de apertura **{** y cierre **}**. Estas llaves delimitan un ámbito –scope- y por lo tanto podemos declarar variables con **let** y que estas variables sólo sean válidas dentro de las llaves.

Otra característica es la capacidad de anidar una estructura dentro de otras, de forma que podemos crear algoritmos complejos con múltiples respuestas.

CONDICIONES

Cuando queremos programar diferentes respuestas u opciones en nuestro código, lo hacemos según una condición: si se cumple, damos una respuesta, si no, damos otra respuesta. Es decir: como programadores debemos dejar en el código todas las opciones que necesitamos plantear, pero en la ejecución de este código sólo se mostrará la opción adecuada en ese momento.

Para plantear las condiciones tenemos diferentes estructuras: el **IF** nos permite plantear una posibilidad y el **ELSE** complementa la contraria; por otra parte, el **SWITCH** nos permite plantear diferentes posibilidades.

IF

La estructura más utilizada en JavaScript y en la mayoría de lenguajes de programación es la estructura **if**. Se emplea para tomar decisiones en función de una condición. Su definición formal es:

```
<script>
    if ( condición ) {
        ...
    }
</script>
```

Si la expresión contenida entre paréntesis o **condición** devuelve un valor ***true*** (por ejemplo, una comparación) se ejecutan todas las instrucciones –sentencias- que se encuentran dentro de {...}. Si la condición no se cumple (es decir, si su valor es ***false***) no se ejecuta ninguna instrucción –sentencias- contenida en {...} y el programa continúa ejecutando el resto de instrucciones –sentencias- del script.

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

```
<script>
    var aceptacion = false, edad = 0, controlEdad = false, controlAcceso = false;
    aceptacion = confirm("Aceptas la política de privacidad?");
    edad = parseInt(prompt("Dime tu edad "));
    controlEdat = edad >= 18
    controlAcceso = aceptacion && controlEdat;
    if ( controlAcceso === true ) {
        alert(`¡Bienvenido/da!`);
    }
    if ( controlAcceso === false ) {
        alert(`No puede acceder`);
    }
</script>
```

La expresión dentro de los paréntesis debe devolver un valor booleano de *true*, y en el ejemplo anterior la variable *controlAcceso* ya es de este tipo, por lo tanto la primera comparación es redundante, y la segunda comparación se puede simplificar si empleamos el operador de negación:

```
<script>
    var aceptacion = false, edad = 0, controlEdad = false, controlAcceso = false;
    aceptacion = confirm("Aceptas la política de privacidad?");
    edad = parseInt(prompt("Dime tu edad "));
    controlEdat = edad >= 18
    controlAcceso = aceptacion && controlEdat;
    if ( controlAcceso ) {
        alert(`¡Bienvenido/da!`);
    }
    if ( !controlAcceso ) {
        alert(`No puede acceder`);
    }
</script>
```

En este ejemplo se plantea las dos posibilidades de la variable *controlAcceso*: que sea *true* o *false*, pero según las respuestas a las preguntas de *aceptacion* y *edad* sólo tendrá un único valor, por lo tanto sólo una de las condiciones se cumplirá y uno de los alert se ejecutará.

IF ... ELSE

Normalmente, las decisiones a realizar no son del tipo "si se cumple la condición, hazlo; si no se cumple, no hagas nada", sino suelen ser del tipo "si se cumple esta condición, hazlo; si no se cumple, haz eso otro".

Para este segundo tipo de decisiones, existe una variante de la estructura **if** llamada **if... else**. Su definición formal es la siguiente:

```
<script>
    if ( condición ) {
        ...
    } else {
        ...
    }
</script>
```

Si la expresión contenida entre paréntesis o **condición** devuelve un valor ***true*** (por ejemplo, una comparación) se ejecutan todas las instrucciones –sentencias- que se encuentran dentro del primer {...}. Si la condición no se cumple (es decir, si su valor es ***false***) se ejecutan todas las instrucciones –sentencias- que se encuentran dentro del segundo {...} precedido de **else**.

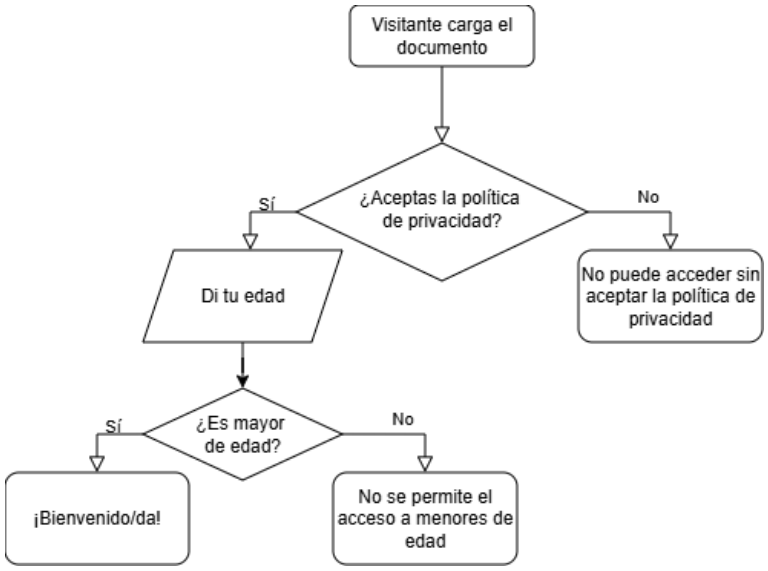
Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

El comando **else** no puede ir solo: siempre acompañará a un **if** para definir la situación contraria o ***false*** de la condición.

```
<script>
    var aceptacion = false, edad = 0, controlEdad = false, controlAcceso = false;
    aceptacion = confirm("Aceptas la política de privacidad?");
    edad = parseInt(prompt("Dime tu edad "));
    controlEdat = edad >= 18
    controlAcceso = aceptacion && controlEdat;
    if ( controlAcceso ) {
        alert(`¡Bienvenido/da!`);
    }
    else {
        alert(`No puede acceder`);
    }
</script>
```

En este ejemplo, hemos sustituido el segundo **if** y su condición por un **else**: en ambas condiciones se evaluaba la misma variable, por lo tanto, podemos simplificar la estructura de control de flujo condicional a una sola evaluación con dos posibilidades.

Como se menciona al principio del capítulo, las estructuras de control se pueden anidar si queremos dar diferentes respuestas. Por ejemplo, según este diagrama queremos dar diferentes respuestas y a continuación se plantea el código con **if** anidados:




```
<script>
    var aceptacion = false, edad = 0;
    aceptacion = confirm("¿Aceptas la política de privacidad?");
    if ( !aceptacion ) {
        alert("No puede acceder sin aceptar la política de privacidad ");
    } else {
        edad = parseInt(prompt("Di tu edad "));
        if ( edad < 18 ) {
            alert("No se permite el acceso a menores de edad");
        } else {
            alert("¡Bienvenido/da!");
        }
    }
</script>
```


Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

ELSE ... IF

Cuando necesitamos verificar varias opciones lo más fácil es utilizar **else if**, así evitamos tener que anidar en exceso. Es importante recordar que la primera condición válida será la que use el navegador, y que ignorará todas las demás. Su definición formal es la siguiente:

```
<script>
    if ( condición ) {
        ...
    } else if ( condición ) {
        ...
    } else if ( condición ) {
        ...
    } else {
        ...
    }
</script>
```

Si la primera condición es *false*, no se ejecuta y salta a la segunda; si ésta también es *false*, no se ejecuta y salta a la tercera; y así hasta que encuentra una condición *true* que se ejecute o el último **else**.

Con el ejemplo anterior, quedaría así el código con **else ... if**:

```
<script>
    var aceptacion = false, edad = 0;
    aceptacion = confirm("¿Aceptas la política de privacidad?");
    edad = parseInt(prompt("Di tu edad "));
    if ( !aceptacion ) {
        alert("No puede acceder sin aceptar la política de privacidad ");
    } else if ( edad < 18 ) {
        alert("No se permite el acceso a menores de edad");
    } else {
        alert("¡Bienvenido/da!");
    }
</script>
```

TERNARIO

Para simplificar la estructura de una condición tenemos el operador condicional ternario: no es un operador como tal sino una estructura donde su definición formal es la siguiente:

```
condicion ? expresionTrue : expresionFalse
```

Es muy práctica en el caso de instanciar una variable con dos posibilidades:

```
<script>
    var edad = parseInt(prompt("Di tu edad "));
    var mensaje = ( edad >= 18 ) ? "¡Bienvenido/da!" : " No se permite el acceso a menores de edad";
    alert(mensaje);
</script>
```

SWITCH

Hasta ahora las estructuras **if** permiten evaluar una condición que puede ser *true* o *false*; pero si la condición puede tener múltiples valores, la estructura **switch** permite evaluar diferentes posibilidades llamadas casos. Cada caso es un punto de entrada, pero hay que definir el final de cada caso con un **break**. Y si se da la situación que ningún caso coincide, podemos establecer un resultado por defecto con el **default**:

```
<script>
```


Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

```
    switch ( expresión ) {
        case valor1:
            sentencias;
            break;
        case valor2:
            sentencias;
            break;
        case valor3:
        case valor4:
            sentencias;
            break;
        default:
            sentencias;
            break;
    }
</script>
```

Como los casos son solo puntos de inicio y no de final, podemos poner diferentes casos seguidos para que tengan el mismo resultado. La expresión puede ser una variable tipo texto o numérica, los valores son las diferentes posibilidades y a continuación utilizamos los dos puntos "**:**" para indicar todas las sentencies que deben ejecutarse hasta el **break**. Sin el **break**, se ejecutarían todas las sentencies hasta el cierre del **switch**.

```
<script>
    const nombreUsuario = prompt("Di tu nombre:");
    var mensaje = "";
    switch ( nombreUsuario ) {
        case "Jordi":
            mensaje = "Eso es pan comido.";
            break;
        case "María":
            mensaje = "De tal palo, tal astilla.";
            break;
        case "Pep":
        case "Ona":
            mensaje = "Más vale prevenir que curar.";
            break;
        default:
            mensaje = "No es oro todo lo que reluce.";
            break;
    }
    alert( mensaje );
</script>
```

isNaN()

Un caso especial es la función gobal `isNaN()`: es una función que devuelve un booleano según el valor no es un número –*true*- o sí es un número –*false*-:

```
<script>
    var edad = Number(prompt("Di tu edad")), mensaje = "";
    if ( isNaN(edad) ) {
        mensaje = "!Hay que introducir una edad válida!";
    } else if ( edad >= 18 ) {
        mensaje = "¡Bienvenido/da!";
    } else {
        mensaje = "No se permite el acceso a menores de edad.";
    }
    alert(mensaje);
</script>
```


Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

BUCLES

El concepto de bucle hace referencia a la repetición de sentencias tantas veces como haga falta sin la necesidad de duplicar líneas de código, por lo tanto, rompemos la linealidad de la ejecución del navegador.

WHILE

Permite repetir las sentencies anidadas en las llaves mientras la condición sea *true*. Esto implica que necesitamos una instanciación inicial, una comparación que mantenga el bucle y una actualización para evitar entrar en un bucle infinito:

```
<script>
    var control = valor; // inicialización
    while ( control == valor ) { // condición
        ...
        control = nuevoValor; // actualización
    }
</script>
```

Un ejemplo es pedir un cierto tipo de dato, y no dejar continuar al usuario hasta que lo introduzca correctamente:

```
<script>
    var edad = parseInt(prompt("Di tu edad")), mensaje = "";
    while ( isNaN(edad) ) {
        alert ("!Hay que introducir una edad válida!");
        edad = parseInt(prompt("Di tu edad"));
    }
    mensaje =(edat>=18)? "¡Bienvenido/da!" : " No se permite el acceso a menores de edad ";
    alert(mensaje);
</script>
```

DO ... WHILE

Si lo que necesitamos es hacer una acción como mínimo una vez, y luego evaluar si hay que repetirla, podemos utilizar la estructura **do ... while** porque justamente hace eso: primero ejecuta el contenido de las llaves y luego evalúa si hay que continuar:

```
<script>
    do {
        ...
    } while ( condición )
</script>
```

En el ejemplo anterior, pedimos dos veces la edad; por lo tanto, podemos simplificar el código pidiéndolo siempre una vez y después evaluar si hay que volver a pedirlo:

```
<script>
    var edad = null, mensaje = "";
    do {
        mensaje =(edad===null)?"Di tu edad":"!Hay que introducir una edad válida!";
        edad = parseInt( prompt(mensaje) );
    } while ( isNaN(edad) )
    mensaje =(edat>=18)? "¡Bienvenido/da!" : " No se permite el acceso a menores de edad ";
    alert(mensaje);
</script>
```


Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

FOR

Este bucle presenta una estructura optimizada para controlar la ejecución de la iteración de manera numérica, es decir, para especificar exactamente cuántas veces queremos que se haga el bucle.

El bucle **for** se divide en tres partes separadas por un punto y coma:

1. **Expresión inicial**: será todo aquello que se ejecutará al iniciarse el bucle. Normalmente la declaración de una variable numérica instanciada con el valor inicial.
2. **Condición**: será evaluada antes de cada iteración. Este es el único parámetro obligatorio y es una condición de comparación con la variable inicial que mantiene el bucle mientras devuelve *true*.
3. **Expresión de actualización**: se ejecutará al final de cada iteración. Hay que aumentar o disminuir el valor de la variable inicial para que la condición llegue un momento que vuelva *false* y evitar entrar en un bucle infinito.

```
<script>
    for ( inicialización; condición; actualización ) {
            ...
    }
    for ( let i = 1; i <= 10; i ++ ) {
        console.log(i);
    }
    for ( let i = 10; i >= 1; i -- ) {
        console.log(i);
    }
</script>
```

Normalmente, la variable que controla los bucles **for** se llama **i**, ya que recuerda a la palabra índice y su nombre tan corto ahorra mucho tiempo y espacio. Si hay que hacer bucles anidados podemos utilizar los nombres de variables **j** o **k**.

La variable inicial se declara dentro de la estructura de control, por lo tanto, tiene un ámbito –scope- local, por lo tanto, es el ejemplo de uso del **let**. Estos mismos ejemplos se pueden hacer con **while**, pero no queda el código tan ordenado como con el **for**:

```
<script>
    var i = 1; // inicialización
    while ( i <= 10 ) { // condición
        console.log(i);
        i ++; // actualización
    }
</script>
```

FOR ... IN I FOR ... OF

Un caso especial de bucles son estas estructuras **for** pensadas para recorrer matrices –arrays-, por lo tanto los veremos más adelante.

SENTENCIAS BREAK Y CONTINUE

La estructura de control **for** es muy sencilla de utilizar, pero tiene el inconveniente de que el número de repeticiones que se realizan sólo se pueden controlar mediante las variables definidas en la zona de actualización del bucle. Las sentencias **break** y **continue** permiten manipular el comportamiento normal de los bucles for para detener el bucle o para saltarse algunas repeticiones. Concretamente, la sentencia **break** permite acabar de manera abrupta un bucle y la sentencia **continue** permitiendo saltarse algunas repeticiones del bucle.

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

FUNCIONES

En programación es muy frecuente que un determinado procedimiento de cálculo definido por un grupo de sentencias deba repetirse varias veces, ya sea en un mismo programa o en otros programas, lo que implica que se deba escribir tantos grupos de aquellas sentencias como veces aparezca este proceso.

La herramienta más potente con la que se cuenta para facilitar, reducir y dividir el trabajo en programación, es escribir aquellos grupos de sentencias una sola y única vez bajo la forma de una **función**.

Un programa es un desarrollo complejo de realizar y por lo tanto es importante que esté bien **estructurado** y también que sea inteligible para las personas. Si un grupo de sentencias realiza una tarea bien definida, entonces puede estar justificado el aislar estas sentencias formando una función, aunque resulte que sólo se la nombre o utilice una vez.

Hasta ahora hemos visto cómo resolver un problema planteando un único algoritmo. Con funciones podemos segmentar un programa en varias partes. Ante un problema, planteamos un algoritmo, este puede constar de pequeños algoritmos.

Una función es un conjunto de sentencias encapsuladas que puede ser utilizado desde diferentes partes de un programa tantas veces como haga falta.

Las funciones de JavaScript son el alma de este lenguaje, por ello se consideran ciudadanos de primera clase, una entidad que soporta todas las operaciones generalmente disponibles para otras entidades: estas operaciones normalmente incluyen ser pasados como argumento, retornados de una función y asignados a una variable.

A partir de ahora veremos la aplicación de los principios de la programación funcional en otros apartados de este manual.

DECLARACIÓN E INVOCACIÓN

Podemos **declarar** las funciones de dos formas: instanciando una variable con una función anónima o utilizando el comando **function** y dando un nombre único siguiendo las mismas directrices al dar un nombre a una variable. La primera diferencia entre una variable y una función es que en una variable almacena datos, y en una función almacena sentencias; la segunda, que al declarar una función usamos los paréntesis "()": en el siguiente apartado explicaremos su uso.

```
<script>
    // declaración en una variable con una función anónima:
    const miFuncion = function () {
        ...
    }
    // declaración con nombre:
    function miFuncion () {
        ...
    }
</script>
```

Una vez declarada la función, todo el código dentro de sus llaves quedará a la espera y el navegador no lo ejecutará hasta que la función sea **invocada** a través de su nombre:

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

```
<script>
    // declaración con nombre:
    function miFuncion () {
        ...
    }
    // invocación a través del nombre:
    miFuncion ();
</script>
```

Dentro de las funciones podemos declarar variables, pero como el contenido se cierra entre llaves "{}", las funciones generan su propio ámbito –scope- y podemos utilizar variables globales (declaradas fuera de las funciones) o variables locales (declaradas dentro de las funciones con **let**).

PARÁMETROS Y ARGUMENTOS

Cuando queremos hacer funciones con un nivel de abstracción realmente alto, tenemos que recurrir al aislamiento. De tal forma que nuestra función no dependa de ciertas variables o datos externos a ella.

Cuando declaramos una función, podemos incluir ciertos **parámetros** entre los paréntesis que actuarán como referencias. Funcionarán internamente igual que variables locales, de tal forma que a la hora de ejecutar la función podremos pasarle ciertos **argumentos** –valores- y así tener funciones con un mayor nivel de abstracción.

Al declarar una función podemos añadir tantos parámetros como necesitamos separados por comas, pero las buenas prácticas de programación lo limitan a tres; si necesitas más, es necesario que desgloses la función en varias más simples. Cada parámetro tendrá un nombre propio, como si fuera una variable, pero al ser local se pueden repetir los mismos nombres entre diferentes funciones. Y en la invocación, pasamos los argumentos –valores- de cada parámetro en el mismo orden y también separado por comas ",":

```
<script>
    // declaración con parámetros
    function miFuncion ( param1, param2, param3) {
        ...
    }
    // invocación con argumentos:
    miFuncion ( arg1, arg2, arg3 );
</script>
```

En el siguiente ejemplo, declaramos una función con un parámetro que se utiliza dentro de la función como una variable local, y la invocamos varias veces: en cada invocación pasamos un valor diferente como argumento:

```
<script>
    function saluda( nombre ) {
        alert("Hola " + nombre);
    }
    saluda ( "Ot" );
    saluda ( "Marta" );
</script>
```

FUNCIONES QUE DEVUELEN UN VALOR

Otro de los puntos fuertes a la hora de plantear estructuras de código modulares y reutilizables, es tener en cuenta el retorno. El retorno nos permite devolver un valor al terminar de ejecutarse la función. Este valor puede ser cualquier tipo de dato de los muchos que tenemos en JavaScript. Para que las funciones sean modulares y

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

reutilizables es necesario que no todas finalicen un proceso, por ejemplo mostrar un *alert*, sino que hagan un proceso y devuelvan un valor, y aquel que haya invocado la función recoja este valor y continúe procesándolo.

Para hacer que una función devuelva un valor, solo hay que utilizar el comando **return** al final del código de la misma:

```
<script>
    function mensaje ( texto ) {
        return "Hola " + texto;
    }
    function saluda ( nombre ) {
        alert ( mensaje (nombre) )
    }
    saluda ( "Ot" );
    saluda ( "Marta" );
</script>
```

ANIDAMIENTO

Dentro de las llaves de las funciones podemos anidar otras estructuras ya vistas como condiciones y bucles, pero también otras funciones que sólo se podrán invocar dentro de la función principal. Esto, por un lado, puede complicar el desarrollo del código, pero por el otro nos da muchas más posibilidades de modularización y reutilización.

```
<script>
    var nombre = null;
    function alerta ( tipo ) {
        let texto = "";
        function controlEdad () {
            let edad = null, mensaje = "";
            do {
                mensaje =(edad===null) ? "Di tu edad" : "¡Hay que introducir una edad válida!";
                edad = parseInt( prompt(mensaje) );
            } while ( isNaN(edad) )
            mensaje=(edad>=18)?"¡Bienvenido/da!":"No se permite el acceso a menores de edad";
            return mensaje;
        }
        function preguntaNombre () {
            if ( nombre === null) {
                nombre = prompt("Cuál es tu nombre?");
            }
            return "Hola " + nombre;
        }
        switch ( tipus ) {
            case "edad":
                text = controlEdad();
                break;
            default:
                text = preguntaNombre();
                break;
        }
        return text;
    }
    alert ( alerta ( "edad") );
    alert ( alerta ( "saludo") );
</script>
```


Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

FAT ARROW

Las funciones **fat arrow** se utilizan para omitir la palabra **function** y simplificar la estructura original de las funciones, pero esta estructura es limitada y no se puede usar en todas las situaciones. Esta simplificación también permite que el **return** sea implícito si no se usan las llaves "{}"; incluso podemos ahorrarnos los paréntesis "()" si tenemos un único parámetro. Después, la invocación se hace como siempre a través del nombre.

Desglose de la función flecha:

Según la definición de una función tradicional tendríamos la siguiente declaración:

```
<script>
    function duplicar (a){
        return a * 2;
    }
</script>
```

Pero con las funciones flecha lo podemos simplificar:

1. Elimina la palabra **function** y coloca la flecha entre el parámetro y la llave de apertura:

```
<script>
    const duplicar = (a) => {
        return a * 2;
    }
</script>
```

2. Saca las llaves del cuerpo y la palabra **return**: el retorno está implicito:

```
<script>
    const duplicar = (a) => a * 2;
</script>
```

3. Suprime los paréntesis de los parámetros si sólo hay un único parámetro:

```
<script>
    const duplicar = a => a * 2;
</script>
```

Una de las razones por las que se introdujeron las funciones flecha fue para eliminar complejidades del ámbito **this** y hacer que la ejecución de funciones sea mucho más intuitiva. En las funciones tradicionales, de manera predeterminada, **this** está en el ámbito del *window* (del documento), pero en las funciones flecha no predeterminan **this** en el ámbito o alcance del documento: lo ejecutan en el ámbito o alcance en que se crean. Este concepto de **this** se desarrolla más adelante en este manual.

FUNCIONES GENERALES

Las funciones generales son funciones ya definidas en el JavaScript; en este manual ya se han comentado algunas como las funciones de conversión de tipo de variable o para evaluar si un valor es o no un número, pero existen algunas más. La más destacable es la función **eval()** que permite evaluar una expresión de texto aportada como argumento como si fuera una sentencia:

```
<script>
    const texto = "2 + 3";
    alert( eval(texto) );
</script>
```


Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

EVENTOS

En la introducción de este manual se explica que los eventos son otra forma de incluir código JavaScript en el código HTML como atributos de las etiquetas: el valor del atributo es el código JavaScript:

```
<button type="button" evento="sentencias;">Botón</button>
```

Los eventos son las diferentes formas que tenemos de interactuar con los diferentes elementos –etiquetas- del código HTML del documento. Una misma etiqueta puede tener asociados diferentes eventos.

Pero dentro de estas comillas estamos muy limitados para añadir todas las estructuras que hemos visto hasta ahora porque no podemos añadir el código en diferentes líneas y el código se ofusca mucho, y tampoco podemos emplear comillas dobles. Por lo tanto, los eventos son una forma ideal de combinar con las funciones: podemos declararlas en los scripts e invocarlas dentro de los eventos:

```
<script>
    const doblar = a => a * 2;
</script>
<p>
    <button type="button" onclick="alert( doblar(3) );">Doblar 3</button>
    <button type="button" onclick="alert( doblar(7) );">Doblar 7</button>
</p>
```

Nombre del evento como atributo	Definición
onblur	Cuando un elemento de formulario pierde el foco
onchange	Cuando el valor de un campo de formulario es modificado
onclick	Cuando se hace clic con el botón del ratón
oncontextmenu	Cuando se hace clic con el botón alternativo del ratón
ondblclick	Cuando se hace doble clic en un objeto
onfocus	Cuando un elemento de formulario adquiere el foco
oninput	Cuando se está modificando un campo de formulario
onkeydown	Cuando se presiona una tecla
onkeypress	Cuando se presiona una tecla
onkeyup	Cuando se deja de presionar una tecla
onload	Cuando una página o imagen acaba de cargarse
onmousedown	Cuando se pite el botón del ratón
onmousemove	Cuando se mueve el ratón
onmouseout	Cuando el cursor del ratón sale del elemento
onmouseover	Cuando el cursor del ratón se pone encima
onmouseup	Cuando se deja ir el botón del ratón
onreset	Cuando se pite el botón de reset de un formulario
onresize	Cuando se modifica el tamaño de una ventana
onselect	Cuando se selecciona texto de un campo de formulario
onsubmit	Cuando se pite el botón sumido de un formulario
onwheel	Cuando la rueda del ratón sube o baja sobre un elemento

En el siguiente ejemplo se ve el evento **onsubmit** y **oninput** aplicado a un formulario:

```
<form action="" onsubmit="return confirm('¿Quieres enviar el formulario?');">
    <input type="text" name="form" value="¡Enviado!" oninput="alert('¡Cambio!');">
    <button type="submit">Enviar</button>
</form>
```


Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

OBJETOS INTEGRADOS DEL LENGUAJE

Son los objetos nativos del lenguaje, definidos por la especificación ECMAScript. Existen siempre, independientemente de donde se ejecute JavaScript. También se llaman tipo nativos (*native objects*) o constructores nativos (*native constructores*). Estos objetos forman parte del núcleo del lenguaje: no los creas tú, ya están cuando arranca el motor de JavaScript.

Como ya están creados, funcionan como plantillas que podemos instanciar en variables con la función constructora **new**. Estos objetos nos proporcionan formas más complejas para manipular la información gracias a que tienen **propiedades** y **métodos**.

Podemos imaginar los objetos del JavaScript como los objetos físicos reales: si cogemos un rotulador, este tiene propiedades –información- como la longitud, volumen o color, pero también tiene métodos –acciones- como destapar, pintar o tapar.

Como tenemos los objetos ya definidos, sus propiedades y métodos también están ya definidos, es decir, tienen nombres ya reservados. Sintácticamente, utilizaremos la función constructora **new** para crear un objeto nativo:

```
<script>
    const texto = new String("¡Hola mundo!");
</script>
```

A partir de ahora, la variable texto será un nuevo objeto (en este caso *string*) y podemos aplicar propiedades y métodos con la sintaxis del punto ".". Los métodos, al ser acciones, son funciones ya predefinidas y se escriben con paréntesis "()" porque pueden tener argumentos:

```
<script>
    const texto = new String("¡Hola mundo!");
    console.log( texto.length ); // 12
    console.log( texto.toUpperCase() ); // "¡HOLA MUNDO!"
</script>
```

TEXTO

El objeto *String* es el que nos permite manipular los textos, pero a nivel de aplicación de métodos y propiedades, una variable tipo texto también los recibe:

```
<script>
    const saludo1 = new String("¡Hola mundo!");
    const saludo2 = "¡Hola mundo!";
</script>
```

Propiedad	Descripción
length	Devuelve la longitud de una cadena.

Método (argumentos)	Descripción
at()	Devuelve un carácter indexado de una cadena.
charAt()	Devuelve el carácter a un índice (posición) especificado.
charCodeAt()	Devuelve el Unicode del carácter en un índice especificado.
codePointAt()	Devuelve el valor Unicode en un índice (posición) de una cadena.

Generalitat de Catalunya
MINISTERIO DE EDUCACIÓN, FORMACIÓN PROFESIONAL Y DEPORTES
MINISTERIO DE TRABAJO Y ECONOMÍA SOCIAL
SERVICIO PÚBLICO DE EMPLEO ESTATAL SEPE SERVEI PÚBLIC D'OCUPACIÓ ESTATAL
Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

concat()	Devuelve dos o más cadenas unidas.
endsWith()	Devuelve si una cadena acaba con un valor especificado.
fromCharCode()	Devuelve los valores Unicode como caracteres.
includes()	Devuelve si una cadena contiene un valor especificado.
indexOf()	Devuelve el índice (posición) de la primera aparición de un valor en una cadena.
isWellFormed()	Retornos ciertos si una cadena está bien formada.
lastIndexOf()	Devuelve el índice (posición) de la última aparición de un valor en una cadena.
localeCompare()	Compara dos cadenas en la localización actual.
match()	Busca una cadena por un valor, o una expresión regular, y devuelve las coincidencias.
matchAll()	Busca una cadena por un valor, o una expresión regular, y devuelve las coincidencias.
padEnd()	Pone una cadena al final.
padStart()	Apade una cadena desde el principio.
prototype	Te permite añadir propiedades y métodos a un objeto.
repeat()	Devuelve una nueva cadena con varias copias de una cadena.
replace()	Busca un patrón en una cadena y devuelve una cadena donde se sustituye la primera coincidencia.
replaceAll()	Busca un patrón en una cadena y devuelve una nueva cadena donde se sustituyen todas las coincidencias.
search()	Busca en una cadena un valor, o expresión regular, y devuelve el índice (posición) de la coincidencia.
slice()	Extrae una parte de una cadena y devuelve una nueva cadena.
split()	Divide una cadena en una matriz de subcadenas.
startsWith()	Comprueba si una cadena empieza con caracteres especificados.
substr()	Depreciado. Utiliza substring() o slice() en lugar de eso.
substring()	Extrae caracteres de una cadena, entre dos índices (posiciones) especificados.
toLocaleLowerCase()	Devuelve una cadena convertida a letras minúsculas, utilizando la localización del anfitrión.
toLocaleUpperCase()	Devuelve una cadena convertida a mayúsculas, utilizando la localización del anfitrión.
toLowerCase()	Devuelve una cadena convertida a letras minúsculas.
toString()	Devuelve una cadena o un objeto cadena como cadena.
toUpperCase()	Devuelve una cadena convertida a letras mayúsculas.
toWellFormed()	Devuelve una cadena donde "sustitutos solitarios" se sustituyen por el carácter de sustitución Unicode.
trim()	Devuelve una cadena con espacios en blanco eliminados.
trimEnd()	Devuelve una cadena con espacios en blanco eliminados desde el final.
trimStart()	Devuelve una cadena con espacios en blanco eliminados desde el inicio.
valueOf()	Devuelve el valor primitivo de una cadena o de un objeto cadena.

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

NÚMERO

El objeto integrado **Math** nos aporta infinidad de recursos matemáticos avanzados como la constante de Euler, gestión de logaritmos, senos, cosenos, tangentes... Cada lector debe indagar y valorar lo que realmente quiere usar, ya que muchos de estos métodos y propiedades van más allá de nuestros objetivos, y no aportan directamente valor al contexto de aprender a programar en JavaScript. Pero sí hay algunos métodos que pueden ser útiles:

Método (argumentos)	Descripción
ceil(x)	Devuelve x, redondeado hacia arriba al entero más cercano.
floor(x)	Devuelve x, redondeado hacia abajo al entero más cercano.
max(x1,x2,..)	Devuelve el número con el valor más alto.
min(x1,x2,..)	Devuelve el número con el valor más bajo.
random()	Devuelve un número aleatorio entre 0 y 1.
round(x)	Redondea x al entero más cercano.

```
<script>
    var x = Math.random() * 10;
    x = Math.ceil(x);
    console.log(x);
</script>
```

El objeto integrado **Number** nos da acceso a métodos similares a las funciones generales:

Método (argumentos)	Descripción
isFinite()	Comprueba si un valor es un número finito.
isInteger()	Comprueba si un valor es un entero.
isNaN()	Comprueba si un valor es NaN.
parseFloat()	Analiza una cadena y devuelve un número.
parseInt()	Analiza una cadena y devuelve un número entero.
toFixed(x)	Formata un número con x números de dígitos después del punto decimal.
toLocaleString()	Convierte un número en una cadena, según la configuración local.
toPrecision(x)	Formata un nombre a x longitud.
toString()	Convierte un número en una cadena.
valueOf()	Devuelve el valor primitivo de un número.

```
<script>
    let x = Math.random() * 10;
    console.log( x.toFixed(0) );
</script>
```

ARRAY

Los arrays son estructuras que nos permiten almacenar muchos datos, sin tener que preocuparnos por el orden o la organización interna: se organiza automáticamente. Otra forma más sencilla de entenderlo, es imaginar que un array es sencillamente como una lista de diferentes valores tipo texto, números, booleanos e incluso otros arrays anidados -llamados arrays multidimensionales-.

Generalitat de Catalunya
MINISTERIO DE EDUCACIÓN, FORMACIÓN PROFESIONAL Y DEPORTES
MINISTERIO DE TRABAJO Y ECONOMÍA SOCIAL
SERVICIO PÚBLICO DE EMPLEO ESTATAL SEPE SERVEI PÚBLIC D'OCUPACIÓ ESTATAL
Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

Podemos instanciar una variable con la cláusula **new** o de forma abreviada con corchetes "**[]**":

```
<script>
    const nombres1 = new Array('María','José');
    const nombres2 = ['María','José'];
</script>
```

Si queremos recuperar algún valor del array, solo hay que utilizar el nombre del objeto y entre los corchetes añadir un número de posición, teniendo en cuenta que los valores de un array empiezan a ordenarse desde el 0:

```
<script>
                    //  0        1
    const nombres = ['María','José'];
    console.log( nombres[0] ); // María
</script>
```

Propiedad	Descripción
length	Devuelve la cantidad de elementos del array.

Con esta propiedad podemos hacer un bucle para recorrer todos los elementos de un array:

```
<script>
    const nombres = ['María','José'];
    const longitud = nombres.length;
    for (let i = 0; i < longitud; i ++) {
        console.log( nombres[i] );
    }
</script>
```

Pero en el captítulo de los bucles se mencionan las estructuras **for ... in** i **for ... of** que son estructuras más optimizadas para reccorer los valores de un array:

```
<script>
    const nombres = ['María','José'];
    for (let i in nombres) { // la variable "i" almacena las posiciones
         console.log( nombres[i] );
    }
    for (let nombre of nombres) { // la variable "nombre" almacena el valor
        console.log( nombre );
    }
</script>
```

Método (argumentos)	Descripción
at()	Devuelve un elemento indexado de un array.
concat()	Une arrays y devuelve un array con los arrays unidos.
copyWithin()	Copia elementos del array dentro del array, hacia y desde posiciones especificadas.
entries()	Devuelve un par clave/valor.
every()	Comprueba si cada elemento de un array supera una prueba.
fill()	Llena los elementos de un array con un valor estático.
filter()	Crea un nuevo array con cada elemento de un array que supera una prueba.
find()	Devuelve el valor del primer elemento de un array que supera una prueba.

Generalitat de Catalunya
MINISTERIO DE EDUCACIÓN, FORMACIÓN PROFESIONAL Y DEPORTES
MINISTERIO DE TRABAJO Y ECONOMÍA SOCIAL
SERVICIO PÚBLICO DE EMPLEO ESTATAL SEPE SERVEI PÚBLIC D'OCUPACIÓ ESTATAL
Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

findIndex()	Devuelve el índice del primer elemento de un array que supera una prueba.
findLast()	Devuelve el valor del último elemento de un array que ha pasado una prueba.
findLastIndex()	Devuelve el índice del último elemento de un array que ha pasado una prueba.
flat()	Concatena elements de subarray.
flatMap()	Mapea todos los elementos del array y crea un nuevo array plano.
forEach()	Llama una función para cada elemento del array.
from()	Crea un array a partir de un objeto.
includes()	Comprueba si un array contiene el elemento especificado.
indexOf()	Busca un elemento en el array y devuelve su posición.
isArray()	Comprueba si un objeto es un array.
join()	Une todos los elementos de un array en una cadena.
keys()	Devuelve un Objeto de Iteración del Array, que contiene las claves del array original.
lastIndexOf()	Busque un elemento en el array, empezando por el final, y devolviendo su posición.
map()	Crea un nuevo array con el resultado de llamar a una función para cada elemento del array.
of()	Crea un array a partir de varios argumentos.
pop()	Elimina el último elemento de un array y devuelve este elemento.
push()	Añade nuevos elementos al final de una matriz y devuelve la nueva longitud.
reduce()	Reduce los valores de un arrebato a un solo valor (de izquierda a derecha).
reduceRight()	Reduce los valores de un array a un solo valor (de derecha a izquierda).
reverse()	Invierte el orden de los elementos de un array.
shift()	Elimina el primer elemento de un array y devuelve este elemento.
slice()	Selecciona una parte de un array y devuelve el nuevo array.
some()	Comprueba si alguno de los elementos de un array supera una prueba.
sort()	Ordena los elementos de un array.
splice()	Añade o elimina elementos del array.
toReversed()	Invierte el orden de los elementos del array (a un nuevo array).
toSorted()	Ordena los elementos de un array (a un nuevo array).
toSpliced()	Añade o elimina elementos del array (a un nuevo array).
toString()	Convierte un array en una cadena y devuelve el resultado.
unshift()	Añade nuevos elementos al inicio de una matriz y devuelve la nueva longitud.
valueOf()	Devuelve el valor primitivo de un array.
with()	Devuelve un nuevo array con elementos actualizados.

```
<script>
    const nombres = ['Maria','Josep'];
    nombres.push('Carles');
    nombres.forEach( (nombre) => {console.log(nombre);} );
</script>
```


Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

FECHA

El objeto integrado **Date** es el que nos permite recuperar información de la fecha del sistema y manipularla. Como ya hemos visto, instanciemos una variable con la cláusula **new** y como argumento le proporcionamos la fecha que nos interese:

```
<script>
    var ahora = new Date(); // fecha actual
    var dia2 = new Date(3600*24*1000); // fecha en milisegundos desde 01/01/1970
    var anoNuevo = new Date("January 1, 2026 00:00:00");//fecha en texto,no recomanable
    var diaAnoNuevo = new Date("2026,1,1"); // fecha en números: AAAA, MM, DD
    var iAnoNuevo = new Date("2026,1,1,0,0,0");//fecha en números: AAAA,MM,DD,HH,MM,SS
</script>
```

En el caso de las fechas, podemos dividir casi todos los métodos en tres categorías principales:

- **Getters**: Que nos devuelvan información concreta.
- **Setters**: Que nos permiten ajustar información concreta.
- **Otros**: Que nos facilitarán enormemente el trabajo para convertir la información.

Método (argumentos)	Descripción
getDate()	Devuelve el día del mes (del 1 al 31)
getDay()	Devuelve el día de la semana (de 0 a 6)
getFullYear()	Devuelve el año
getHours()	Devuelve la hora (de 0 a 23)
getMilliseconds()	Devuelve los milisegundos (de 0 a 999)
getMinutes()	Devuelve los minutos (de 0 a 59)
getMonth()	Devuelve el mes (de 0 a 11)
getSeconds()	Devuelve los segundos (de 0 a 59)
getTime()	Devuelve el número de milisegundos desde medianoche del 1 de enero de 1970 y una fecha especificada
getTimezoneOffset()	Devuelve la diferencia horaria entre la hora UTC y la hora local, en minutos
getUTCDate()	Devuelve el día del mes, según la hora universal (del 1 al 31)
getUTCDay()	Devuelve el día de la semana, según la hora universal (de 0 a 6)
getUTCFullYear()	Devuelve el año, según el tiempo universal
getUTCHours()	Devuelve la hora, según la hora universal (de 0 a 23)
getUTCMilliseconds()	Devuelve los milisegundos, según el tiempo universal (de 0 a 999)
getUTCMinutes()	Devuelve los minutos, según el tiempo universal (de 0 a 59)
getUTCMonth()	Devuelve el mes, según el tiempo universal (de 0 a 11)
getUTCSeconds()	Devuelve los segundos, según el tiempo universal (de 0 a 59)
getYear()	Depreciado. Utiliza el método getFullYear() en cambio
now()	Devuelve el número de milisegundos desde medianoche del 1 de enero de 1970
parse()	Analiza una cadena de fechas y devuelve el número de milisegundos desde el 1 de enero de 1970
setDate()	Establece el día del mes de un objeto de fecha
setFullYear()	Establece el año de un objeto de fecha

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

setHours()	Establece la hora de un objeto de fecha
setMilliseconds()	Establece los milisegundos de un objeto de fecha
setMinutes()	Establece las actas de un objeto de fecha
setMonth()	Establece el mes de un objeto de fecha
setSeconds()	Establece los segundos de un objeto de fecha
setTime()	Fija una fecha en un número especificado de milisegundos después o antes del 1 de enero de 1970
setUTCDate()	Fija el día del mes de un objeto de fecha, según el tiempo universal
setUTCFullYear()	Establece el año de un objeto de fecha, según el tiempo universal
setUTCHours()	Fija la hora de un objeto de fecha, según el tiempo universal
setUTCMilliseconds()	Establece los milisegundos de un objeto de fecha, según el tiempo universal
setUTCMinutes()	Establece los minutos de un objeto de fecha, según la hora universal
setUTCMonth()	Fija el mes de un objeto de fecha, según el tiempo universal
setUTCSeconds()	Fija los segundos de un objeto de fecha, según el tiempo universal
setYear()	Depreciado. Utiliza el método setFullYear() en cambio
toDateString()	Convierte la parte de fecha de un objeto Date en una cadena legible
toGMTString()	Depreciado. Utiliza el método toUTCString() en cambio
toISOString()	Devuelve la fecha como cadena, utilizando el estándar ISO
toJSON()	Devuelve la fecha como cadena, formatada como fecha JSON
toLocaleDateString()	Devuelve la parte de fecha de un objeto Date como cadena, utilizando convenciones locales
toLocaleTimeString()	Devuelve la parte de tiempo de un objeto Date como cadena, utilizando convenciones locales
toLocaleString()	Convierte un objeto Date en una cadena, utilizando convenciones de localización
toString()	Convierte un objeto Date en una cadena
toTimeString()	Convierte la parte temporal de un objeto Date en una cadena
toUTCString()	Convierte un objeto Date en una cadena, según el tiempo universal
UTC()	Devuelve el número de milisegundos en una fecha desde medianoche del 1 de enero de 1970, según la hora UTC
valueOf()	Devuelve el valor primitivo de un objeto Date

En siguiente ejemplo se aplican el método **toLocaleDateString()** para quedarnos sólo con la fecha con un formato estándar, o los métodos **getDay()**, **getDate()**, **getMonth()** y **getFullYear()** para recuperar cada una de las informaciones de la fecha actual para darle el formato que queramos. Como estos métodos vuelven un número, utilizamos matrices con los nombres personalizados para convertir los números a texto. Finalmente, utilizamos los métodos **getHours()**, **getMinutes()** y **getSeconds()** para recuperar la información de la hora, con condiciones ternarias para añadir un 0 inicial si el valor es inferior a 10:

```
<script>
    const d = new Date();

    console.log( d.toLocaleDateString() );
    // 10/1/2026

    const diasSemana = ['Domingo','Lunes', 'Martes', 'Miércoles', 'Jueves',
'Viernes', 'Sábado'];
```


Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

```
    const mesesAno = ['enero', 'febrero', 'marzo', 'abril', 'mayo', 'junio',
'julio', 'agosto', 'septiembre', 'octubre', 'noviembre', 'diciembre'];

    console.log( diasSemana [d.getDay()] + " " + d.getDate() + " de " + mesesAno
[d.getMonth()] + " del " + d.getFullYear() );
    // Sábado 10 de enero del 2026

    console.log(`${(d.getHours()<10)? "0"+d.getHours() : d.getHours()} :
${(d.getMinutes()<10)? "0"+d.getMinutes() : d.getMinutes()} :
${(d.getSeconds()<10)? "0"+d.getSeconds() : d.getSeconds()}`);
    // 09 : 08 : 38
</script>
```

OBJECTES HOST

Son objetos que no forman parte del lenguaje, sino que les proporciona el entorno donde se ejecuta JavaScript; en nuestro caso, el navegador y el mismo documento HTML.

BOM

El **Browser Object Model** es la forma que interpreta el JavaScript el navegador como un objeto: de esta manera le puede aplicar propiedades y métodos. El objeto **Window** tiene una serie de propiedades como **Console**, **History**, **Location**, **Navigator** o **Screen**, y a su vez estas propiedades tienen sus propios métodos:

Propiedades y métodos WINDOW	Descripción
window.addEventListener()	Adjunta un gestor de eventos a una ventana
window.alert() o alert()	Muestra una caja de alerta con un mensaje y un botón de aceptación
window.confirm() o confirm()	Muestra un cuadro de diálogo con un mensaje, un botón de OK y un botón de Cancel
window.prompt() o prompt()	Muestra un cuadro de diálogo que pide al usuario la entrada de texto
window.open()	Abre una nueva ventana del navegador o una nueva pestaña, dependiendo de la configuración del navegador y de los valores de los parámetros
setInterval()	Llama una función a intervalos específicos (en milisegundos).
clearInterval()	Borra un temporizador establecido con el método setInterval()
setTimeout()	Llama una función después de un número de milisegundos
clearTimeout()	Borra un temporizador establecido con el método setTimeout()
window.innerHeight	Devuelve la altura del área de contenido de una ventana
window.innerWidth	Devuelve la anchura del área de contenido de una ventana
window.outerHeight	Devuelve la altura exterior de la ventana del navegador, incluyendo todos los elementos de la interfaz (como las barras de herramientas o las barras de desplazamiento).
window.outerWidth	Devuelve la anchura exterior de la ventana del navegador, incluyendo todos los elementos de la interfaz (como las barras de herramientas o las barras de desplazamiento)
window.scrollBy()	Desplaza el documento por el número especificado de píxeles
window.scrollTo()	Desplaza el documento hasta las coordenadas especificadas
window.scrollX **window.scrollY** **window.pageXOffset** **window.pageYOffset**	Devuelve los píxeles que un documento ha desplazado desde la esquina superior izquierda de la ventana
window.print()	Abre el cuadro de diálogo de impresión, que permite al usuario seleccionar opciones de impresión preferidas

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

window.localStorage o localStorage	El objeto localStorage almacena datos sin fecha de caducidad. Los datos no se eliminan y están disponibles para futuras sesiones.
localStorage.setItem()	Te permite desartar pares clave/valor en el navegador
localStorage.getItem()	Te permite recuperarel valor de una clave
window.console o console	Proporciona acceso a la consola de depuración del navegador
console.log()	Envía un mensaje a la consola
console.table()	Muestra los datos tabulares como una tabla
window.location o location	Contiene información sobre la URL actual
location.hash	Establece o devuelve la parte de anclaje (#) de una URL
location.host	Establece o devuelve el nombre de anfitrión y el número de puerto de una URL
location.hostname	Establece o devuelve el nombre de host de una URL
location.href	Establece o devuelve toda la URL
location.origin	Devuelve el protocolo, el nombre de anfitrión y el número de puerto de una URL
location.pathname	Establece o devuelve el nombre del camino de una URL
location.port	Establece o devuelve el número de puerto de una URL
location.protocol	Establece o devuelve el protocolo de una URL
location.search	Establece o devuelve la parte de la cadena de consulta de una URL
window.history o history	Contiene las URL visitadas por el usuario (en la ventana del navegador).
history.length	Devuelve el número de URLs (páginas) a la lista de historial
history.back()	Carga la URL anterior (página) en la lista de historial
history.forward()	Carga la siguiente URL (página) en la lista de historial
history.go()	Carga una URL específica (página) de la lista de historial
window.navigator o navigator	Contiene información sobre el navegador
navigator.language	Devuelve el lenguaje del navegador
navigator.userAgent	Devuelve la cabecera de usuario-agente enviada por el navegador al servidor.
window.screen o screen	El objeto pantalla contiene información sobre la pantalla del visitante
screen.availHeight	Devuelve la altura de la pantalla (excluyendo la barra de tareas)
screen.availWidth	Devuelve la anchura de la pantalla (excepto la barra de tareas)
screen.height	Devuelve la altura total de la pantalla
screen.width	Devuelve la anchura total de la pantalla

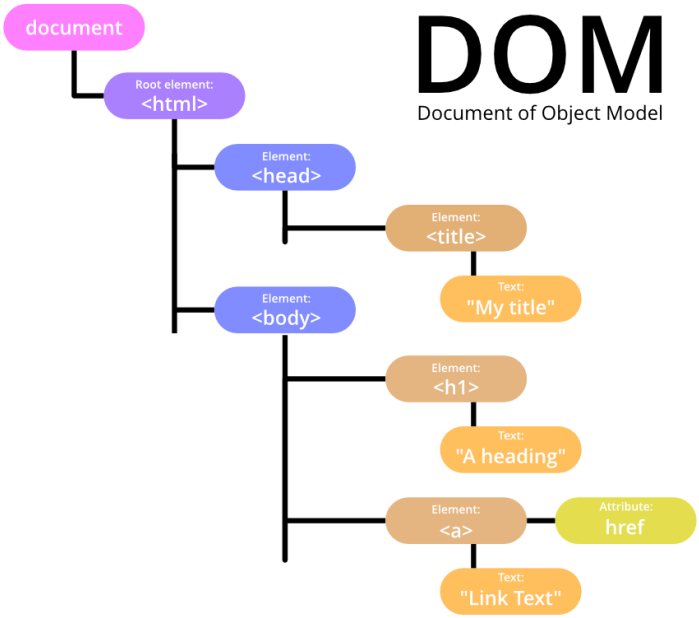
```
<script>
    var elBanner;
    window.addEventListener("load",function(){
        elBanner = setInterval(function(){
            alert("¿Todavía estás aquí?");
        },5000);
    });
    function cerrarBanner () {
        clearInterval(elBanner);
    }
</script>
<p>
    <button type="button" onclick="cerrarBanner();">Parar banner</button>
</p>
```

DOM

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

El **Documento Object Model** es la forma que interpreta el JavaScript el documento como un objeto: de esta manera le puede aplicar propiedades y métodos. El DOM da una representación de las etiquetas HTML del documento como un grupo de **nodos** y objetos estructurados que tienen propiedades y métodos. Esencialmente, conecta las páginas web a scripts o lenguajes de programación.

En el siguiente esquema, cada rectángulo representa un nodo DOM y las flechas indican las relaciones entre nodos. Dentro de cada nodo, se ha incluido su tipo y su contenido.



Documento: nodo raíz del cual derivan todos los demás nodos del árbol.

Elemento: representa cada una de las etiquetas HTML. Se trata del único nodo que puede contener atributos y el único del que pueden derivar otros nodos.

Attributo: se define un nodo de este tipo para representar cada uno de los atributos de las etiquetas HTML, es decir, uno por cada par atributo="valor".

Texto: nodo que contiene el texto cerrado por una etiqueta HTML.

Las funciones que proporciona DOM para acceder a un nodo a través de sus nodos padre consisten en acceder al nodo a raíz de la página y luego a sus nodos hijos y a los nodos hijos de estos hijos y así sucesivamente hasta el último nodo de la rama terminada por el nodo buscado. Sin embargo, cuando se quiere acceder a un nodo específico, es mucho más rápido acceder directamente a este nodo y no llegar hasta él descendiendo a través de todos sus nodos padre.

Por este motivo, no se presentarán las funciones necesarias para el acceso jerárquico de nodos y se muestran solamente las propiedades y métodos del objeto **document** que permiten acceder de manera directa a los nodos:

Propiedad	Descripción
cookie	Devuelve todos los pares de galletas nombre/valor del documento
forms	Devuelve una colección de todos <form> los elementos del documento
images	Devuelve una colección de todos <img> los elementos del documento
links	Devuelve una colección de todos <a> <area> los y elementos del documento que tienen un atributo href
title	Establece o devuelve el título del documento
URL	Devuelve la URL completa del documento HTML

Método (argumentos)	Descripción

addEventListener()	Adjunta un gestor de eventos al documento
getElementById()	Devuelve el elemento que tiene el atributo ID con el valor especificado
getElementsByClassName()	Devuelve un HTMLCollection que contiene todos los elementos con el nombre de clase especificado
getElementsByName()	Devuelve una NodeList activa que contiene todos los elementos con el nombre especificado
getElementsByTagName()	Devuelve un HTMLCollection que contiene todos los elementos con el nombre de etiqueta especificado
hasFocus()	Devuelve un valor booleano que indica si el documento tiene foco
querySelector()	Devuelve el primer elemento que coincide con un selector CSS especificado del documento
querySelectorAll()	Devuelve una NodeList estática que contiene todos los elementos que coinciden con un(s) selector(s) CSS especificado(s) del documento
removeEventListener()	Elimina un gestor de eventos del documento (que se ha adjuntado con el método addEventListener())
write()	Escribe expresiones HTML o código JavaScript en un documento
writeln()	Igual que write(), pero añade un carácter de nueva línea después de cada instrucción

```
<script>
    window.addEventListener("load",function(){
        const elBoton = document.getElementById("btnBoton");
        elBoton.addEventListener("click",function(){
            alert("Has pulsado el botón");
        });
    });
</script>
<p>
    <button type="button" id="btnBoton">Botón</button>
</p>
```

Una vez tenemos seleccionado uno o un conjunto de nodos, podemos aplicar nuevas propiedades y métodos:

Propiedad	Descripción
length	Obtiene el número de nodos seleccionados
classList	Devuelve el(s) nombre(s) de clase de un elemento
clientHeight	Devuelve la altura de un elemento, incluyendo el relleno
clientLeft	Devuelve la anchura del borde izquierdo de un elemento
clientTop	Devuelve la anchura del borde superior de un elemento
clientWidth	Devuelve la anchura de un elemento, incluyendo el relleno
innerHTML	Establece o devuelve el contenido de un elemento
innerText	Establece o devuelve el contenido textual de un nodo y sus descendientes
scrollHeight	Devuelve toda la altura de un elemento, incluyendo el relleno
scrollLeft	Establece o devuelve el número de píxeles en los que el contenido de un elemento está desplazado horizontalmente
scrollTop	Establece o devuelve el número de píxeles en los que el contenido de un elemento se desplaza verticalmente
scrollWidth	Devuelve toda la anchura de un elemento, incluyendo el relleno
style	Establece o devuelve el valor del atributo de estilo de un elemento

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

textContent	Establece o devuelve el contenido textual de un nodo y sus descendientes
(nombreAtributo)	Cualquier atributo de una etiqueta HTML se convierte en una propiedad

Método (argumentos)	Descripción
addEventListener()	Adjunta un gestor de eventos a un elemento
checkValidity()	Comprueba si el elemento tiene restricciones y si las cumple. Si el elemento no cumple sus restricciones, devuelve falso.
hasAttribute()	Devuelve cierto si un elemento tiene un atributo dado
hasAttributes()	Devuelve cierto si un elemento tiene cualquier atributo
hasChildNodes()	Devuelve cierto si un elemento tiene nodos hijos
scrollIntoView()	Desplaza el elemento al área visible de la ventana del navegador
setAttribute()	Establece o cambia el valor de un atributo
setAttributeNode()	Establece o cambia un nodo de atributo
reset()	Restablece un formulario a su estado inicial
submit()	Envía los datos de un formulario
preventDefault()	Cancela el evento si es cancelable, es decir, que la acción por defecto que pertenece al evento no se producirá
stopPropagation()	Impide que se llame la propagación del mismo acontecimiento

Un caso especial es la propiedad **classList** que hace referencia a las clases CSS que se aplican a una etiqueta: como esto implica manipular la estética del nodo, tenemos disponibles más métodos y propiedades especiales sólo para manipular los nombres de las clases:

Propiedades y métodos de classList	Descripción
add()	Añade uno o más nombres a la lista
contains()	Regresa verdadero si la lista contiene una clase
forEach()	Ejecuta una función de llamada para cada nombre de la lista
length	Devuelve el número de nombres a la lista
remove()	Elimina uno o más nombres de la lista
replace()	Sustituye un nombre en la lista
toggle()	Cambia entre fichas en la lista
value	Devuelve la lista de nombres como una cadena
values()	Devuelve un iterador con los valores de la lista

En el siguiente ejemplo añadimos un gestor de evento al objeto window para esperar a que se cargue el documento y que tengamos disponibles los botones. A continuación se seleccionan todos por su nombre de clase CSS, y a cada uno añadimos un gestor de evento para detectar cuando se pegan para mostrar una alerta personalizada con el contenido del botón:

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

```
<script>
    window.addEventListener("load",function(){
        const elBoton = document.querySelectorAll(".elBoton");
        elBoton.forEach(function(b){
            b.addEventListener("click",function(){
                alert("Has pulsado el botón " + b.textContent);
            });
        });
    });
</script>
<p>
    <button type="button" class="elBoton">Botón 1</button>
    <button type="button" class="elBoton">Botón 2</button>
    <button type="button" class="elBoton">Botón 3</button>
</p>
```

Con el método addEventListener() ya no hay que añadir los eventos como atributos de las etiquetas HTML, por lo tanto el código queda mucho más limpio.

Este método necesita como primer argumento el acontecimiento a escuchar: el nombre del acontecimiento es el mismo que el del atributo HTML pero sin el "**on**" inicial:

Nombre del evento como argumento	Definición
blur	Cuando un elemento de formulario pierde el foco
change	Cuando el valor de un campo de formulario es modificado
click	Cuando se hace clic con el botón del ratón
contextmenu	Cuando se hace clic con el botón alternativo del ratón
dblclick	Cuando se hace doble clic en un objeto
focus	Cuando un elemento de formulario adquiere el foco
input	Cuando se está modificando un campo de formulario
keydown	Cuando se presiona una tecla
keypress	Cuando se presiona una tecla
keyup	Cuando se deja de presionar una tecla
load	Cuando una página o imagen acaba de cargarse
mousedown	Cuando se pite el botón del ratón
mousemove	Cuando se mueve el ratón
mouseout	Cuando el cursor del ratón sale del elemento
mouseover	Cuando el cursor del ratón se pone encima
mouseup	Cuando se deja ir el botón del ratón
reset	Cuando se pite el botón de reset de un formulario
resize	Cuando se modifica el tamaño de una ventana
select	Cuando se selecciona texto de un campo de formulario
submit	Cuando se pite el botón sumido de un formulario
wheel	Cuando la rueda del ratón sube o baja sobre un elemento

En el siguiente ejemplo se utilizan diferentes métodos para controlar el envío de un formulario validándolo con JavaScript y no por el navegador:

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

```
<script>
    window.addEventListener("load",function(){
        const forms = document.querySelectorAll('.para-validar');
        forms.forEach(form => {
            form.addEventListener("submit", event => {
                event.preventDefault();
                event.stopPropagation();
                if ( form.checkValidity() ) {
                    form.submit();
                } else {
                    alert("Verifica los campos del formulario");
                }
            });
        });
    });
</script>
<form class="para-validar" novalidate>
    <p>
        <label for="idNom">Nombre</label>
        <input type="text" id="idNombre" required name="elNombre">
    </p>
    <p>
        <button type="submit">Enviar</button>
    </p>
</form>
```

A partir de ahora, con la manipulación del DOM y acceso a los diferentes nodos del documento, podemos abandonar los métodos del objeto **window** como `alert()`, `confirm()` y `prompt()` para emplear elementos propios del lenguaje HTML como párrafos, divisores, botones y entradas de formulario para interactuar con los visitantes y mejorar la experiencia de usuario.

En el siguiente ejemplo, se emplea un formulario para que el visitante introduzca los datos, se manipula la propiedad "value" y "checked" para recoger la información y se muestra la respuesta de forma asíncrona en un contenedor:

```
<script>
    window.addEventListener("load",function(){
        const formAcceso = document.getElementById('formAcceso');
        const mensajeAcceso = document.getElementById('mensajeAcceso');

        formAcceso.addEventListener("submit", event => {
            event.preventDefault();
            event.stopPropagation();
            mensajeAcceso.textContent = "";

            let edad = document.getElementById('edad').value;
            let edadNum = parseInt(edad);
            let politicaLegal = document.getElementById('politicaLegal').checked;
            console.log (edad, edadNum, politicaLegal);

            if (politicaLegal && !isNaN(edadNum) && edadNum >= 18 ) {
                formAcceso.submit();
            } else if ( isNaN(edadNum) ) {
                mensajeAcceso.textContent = "Hay que introducir la edad.";
            } else if ( edadNum < 18 ) {
                mensajeAcceso.textContent = "No se permite el acceso a menores de edad.";
            } else if ( !politicaLegal ) {
                mensajeAcceso.textContent = " Hay que aceptar la política de privacidad.";
            } else {
                mensajeAcceso.textContent = "Llena los campos del formulario.";
```


Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

```
            }
        });

    });
</script>
<form id="formAcceso">
    <p>
        <label for="edad">Introduce tu edad:</label>
        <input type="text" id="edad" name="edad">
    </p>
    <p>
        <input type="checkbox" id="politicaLegal" name="politicaLegal">
        <label for="politicaLegal">Acepto la política de privacitat</label>
    </p>
    <p>
        <button type="submit">Validar</button>
    </p>
</form>
<p id="mensajeAcceso"></p>
```

OBJETOS LITERALES

Un objeto literal es un conjunto de claves y valores. Cada clave es un nombre, y cada valor puede ser lo que quieras: un número, un texto, otro objeto, una función... Es como decir: "*Tengo algo que tiene estas características y puede hacer estas acciones*".

Un objeto en JavaScript es un contenedor flexible que agrupa datos y comportamientos bajo un mismo nombre.

EL OBJETO es una entidad de la vidal real que se traslada al paradigma informático: tienen ATRIBUTOS como características que lo pueden definir y tienen MÉTODOS que son acciones que pueden realizar los objetos sobre sus propios atributos o sobre los de otros objetos. Por ejemplo: el objeto coche tiene propiedades: "color", "marca", "modelo", "motor"; también tiene funciones: arrancar() o frenar().

Una forma rápida de crear un objeto es instanciar una variable con las llaves "{}" y añadir dentro las claves de datos que necesitamos con los valores correspondientes empleando el operador dos puntos ":" y separando las claves con coma ",". Los métodos se definen como una clave más pero con una función anónima como valor. Una vez declarado y instanciado el objeto, podemos hacerle referencia por el nombre y utilizar las propiedades y los métodos con el operador del punto ".":

```
<script>
    const coche = {
        color:'negro',
        marca:'js',
        model:'object',
        motor:'híbrido',
        arrancar: function () {console.log("arrancar");},
        frenar: function () {console.log("frenar");}
    }
    coche.marca = "JavaScript";
    console.log(coche.marca);
    coche.arrancar();
    coche.frenar();
</script>
```

Los valores que podemos almacenar pueden ser de cualquier de los tipos vistos con las variables.

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

Las CLASES son plantillas que definen qué atributos y métodos deben tener todos los objetos creados a partir de esta clase; no asigna valores, sólo los tipos de los mismos. Las clases también pueden tener herencias: de esta forma las clases principales son Superclases, y las subordinadas son inferiores y se llaman Subclases.

Utilizamos **class** para declarar un nuevo objeto (los objetos empiezan siempre con mayúscula y en singular), y **extends** si esta es una subclase de una clase previamente declarada.

La instanciación es la acción de crear un objeto a partir de una clase dentro de una variable y lo hacemos con **new**. En el siguiente ejemplo simularemos una fábrica de bollería y pasteles:

```
<script>
    class Brioixeria {
        // Atributos con los valores por defecto para definir el tipo:
        nom         = '';
        sabor       = '';
        pes         = 0;
        color       = '';
        racions     = 0;
    }
    class Pastis extends Brioixeria {
        // Atributos con los valores por defecto para definir el tipo:
        // Como este objeto depende de una superclase, los atributos originales no hay que llamarlos
        espelmes    = 0;
    }
    const article1 = new Brioixeria();
    article1.nom          = 'Croissant';
    article1.sabor        = 'mantequilla';
    article1.pes          = 0.15;
    article1.color        = 'beige';
    article1.racions      = 1;

    document.writeln(`<dl>
        <dt>Nombre:</dt><dd>${article1.nom}</dd>
        <dt>Sabor:</dt><dd>${article1.sabor}</dd>
        <dt>Peso:</dt><dd>${article1.pes} Kg</dd>
        <dt>Color:</dt><dd>${article1.color}</dd>
        <dt>Raciones</dt><dd>${article1.racions}</dd>
    </dl>`);

    const article2 = new Pastis();
    article2.nom          = 'Pastel de manzana';
    article2.sabor        = 'manzana y caramelo';
    article2.pes          = 1.5;
    article2.color        = 'marrón';
    article2.racions      = 6;
    article2.espelmes     = 12;

    document.writeln(`<dl>
        <dt>Nombre:</dt><dd>${article2.nom}</dd>
        <dt>Sabor:</dt><dd>${article2.sabor}</dd>
        <dt>Peso:</dt><dd>${article2.pes} Kg</dd>
        <dt>Color:</dt><dd>${article2.color}</dd>
        <dt>Raciones</dt><dd>${article2.racions}</dd>
        <dt>Velas</dt><dd>${article2.espelmes}</dd>
    </dl>`);
</script>
```

En este ejemplo tenemos una Superclase Brioixeria que tiene una serie de atributos asignados. Y de esta se crea una Subclase Pastis que hereda sus atributos, pero se pueden añadir nuevos atributos para personalizar el nuevo objeto.

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

Si el objeto tiene muchas propiedades asignadas, instanciar cada una de ellas individualmente con el operador del punto es muy pesado. La opción para agilizar la asignación inicial de valores es crear un constructor o función constructora: dentro de la clase, utilizamos **constructor** como nombre de método genérico que permite añadir valores a los atributos en el mismo momento de la instanciación. Utilizamos **this.** en las propiedades para centrar el ámbito -scope -dentro del objeto y así llamamos sólo a sus atributos, pero si el objeto es una Subclase y hereda las propietas de su Superclase, utilizamos **super.** si hacemos referencia a elementos heredados:

```
<script>
    class Brioixeria {
        nom         = '';
        sabor       = '';
        pes         = 0;
        color       = '';
        racions     = 0;

        constructor (nom, sabor, pes, color, racions) {
            this.nom = nom;
            this.sabor = sabor;
            this.pes = pes;
            this.color = color;
            this.racions = racions;
        }
    }
    class Pastis extends Brioixeria {
        espelmes    = 0;

        constructor (nom, sabor, pes, color, racions, espelmes) {
            super (nom, sabor, pes, color, racions);
            this.espelmes = espelmes;
        }
    }
    const article1 = new Brioixeria('Croissant', 'mantequilla', 0.15, 'beige', 1);

    document.writeln(`<dl>
        <dt>Nombre:</dt><dd>${article1.nom}</dd>
        <dt>Sabor:</dt><dd>${article1.sabor}</dd>
        <dt>Peso:</dt><dd>${article1.pes} Kg</dd>
        <dt>Color:</dt><dd>${article1.color}</dd>
        <dt>Raciones</dt><dd>${article1.racions}</dd>
    </dl>`);

    const article2 = new Pastis('Pastel de manzana', ' manzana y caramelo', 1.5,
'marrón', 6, 12);

    document.writeln(`<dl>
        <dt>Nombre:</dt><dd>${article2.nom}</dd>
        <dt>Sabor:</dt><dd>${article2.sabor}</dd>
        <dt>Peso:</dt><dd>${article2.pes} Kg</dd>
        <dt>Color:</dt><dd>${article2.color}</dd>
        <dt>Raciones</dt><dd>${article2.racions}</dd>
        <dt>Velas</dt><dd>${article2.espelmes}</dd>
    </dl>`);
```

Además del método por defecto de constructor, podemos añadir métodos o acciones personalizadas a nuestros objetos declarando funciones dentro del propio objeto y modificándolo en las Subclases:

```
<script>
    class Brioixeria {
        nom         = '';
        sabor       = '';
```


Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

```
        pes         = 0;
        color       = '';
        racions     = 0;

        constructor (nom, sabor, pes, color, racions) {
            this.nom = nom;
            this.sabor = sabor;
            this.pes = pes;
            this.color = color;
            this.racions = racions;
        }

        aTexto () {
            return `Soy un ${this.nom} de sabor de ${this.sabor} con un peso de
${this.pes} Kg, de color ${this.color} para ${this.racions} raciones`;
        }
    }

    class Pastis extends Brioixeria {
        espelmes    = 0;

        constructor (nom, sabor, pes, color, racions, espelmes) {
            super (nom, sabor, pes, color, racions);
            this.espelmes = espelmes;
        }

        aTexto () {
            return `Soy un ${this.nom} de sabor de ${this.sabor} con un peso de
${this.pes} Kg, de color ${this.color} para ${this.racions} raciones con
${this.espelmes} velas`;
        }
    }

    const article1 = new Brioixeria('Croissant', 'mantequilla', 0.15, 'beige', 1);
    document.writeln( "<p>" + article1.aTexto() + "</p>");
    const article2 = new Pastis('Pastel de manzana', ' manzana y caramelo', 1.5,
'marrón', 6, 12);
    document.writeln( "<p>" + article2.aTexto() + "</p>");
</script>
```

Aunque las propiedades se pueden recuperar y volver a instanciarse en cualquier momento porque JavaScript es un lenguaje que da mucha flexibilidad, en otros lenguajes orientados a objetos, las propiedades se definen como privadas y no se pueden utilizar tan libremente desde fuera de la propia definición del objeto, de ahí que se definan métodos **getters** y **setters** para poder manipularlas:

GETTERS

Los Getters o micrométodos son métodos para poder leer los valores de los atributos, necesarios en otros lenguajes orientados a objetos porque son privados y no se pueden consultar desde fuera del objeto. Como el resto de métodos, son heredables y sólo hay que definirlos una vez en la Superclase con el nombre que queramos. En nuestro ejemplo:

```
    getNom () {
        return this.nom;
    }
    getSabor () {
        return this.sabor;
    }
    getPes () {
```


Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

```
        return this.pes;
    }
    getColor() {
        return this.color;
    }
    getRacions () {
        return this.racions;
    }
```

Y en la Subclase:

```
    getEspelmes () {
        return this.espelmes;
    }
```

SETTERS

Los Setters o micrométodos son métodos para poder modificar los valores de los atributos, necesarios en otros lenguajes orientados a objetos porque son privados y no se pueden modificar desde fuera del objeto. Como el resto de métodos, son heredables y sólo hay que definirlos una vez en la Superclase con el nombre que queramos. En nuestro ejemplo:

```
    setNom (v) {
        this.nom = v;
    }
    setSabor (v) {
        this.sabor = v;
    }
    setPes (v) {
        this.pes = v;
    }
    setColor(v) {
        this.color = v;
    }
    setRacions (v) {
        this.racions = v;
    }
```

Y en la Subclase:

```
    setEspelmes (v) {
        this.espelmes = v;
    }
```

La incorporación de estos micrométodos haría modificar los otros métodos para incorporarlos. En nuestro ejemplo, modificaríamos el método para muestra la información en la Subclase para leer los atributos. Al ser atributos de la Superclase, utilizo **super** y los getters para leer la información; sólo se usa el **this** para los atributos propios:

```
    aTexto () {
        return ` Soy un pastel con el nombre ${super.getNom()} de sabor de
${super.getSabor()} con un peso de ${super.getPes()} Kg, de color
${super.getColor()} para ${super.getRacions()} raciones con ${this.espelmes}
velas`;
    }
```

Ahora, todo junto, el ejemplo queda:

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

```
<script>
    class Brioixeria {
        nom         = '';
        sabor       = '';
        pes         = 0;
        color       = '';
        racions     = 0;

        constructor (nom, sabor, pes, color, racions) {
            this.nom = nom;
            this.sabor = sabor;
            this.pes = pes;
            this.color = color;
            this.racions = racions;
        }

        getNom () {
            return this.nom;
        }
        getSabor () {
            return this.sabor;
        }
        getPes () {
            return this.pes;
        }
        getColor() {
            return this.color;
        }
        getRacions () {
            return this.racions;
        }

        setNom (v) {
            this.nom = v;
        }
        setSabor (v) {
            this.sabor = v;
        }
        setPes (v) {
            this.pes = v;
        }
        setColor(v) {
            this.color = v;
        }
        setRacions (v) {
            this.racions = v;
        }

        aTexto () {
            return `Soy un ${this.nom} de sabor de ${this.sabor} con un peso de
              ${this.pes} Kg, de color ${this.color} para ${this.racions} raciones`;
        }
    }

    class Pastis extends Brioixeria {
        espelmes    = 0;

        constructor (nom, sabor, pes, color, racions, espelmes) {
            super (nom, sabor, pes, color, racions);
            this.espelmes = espelmes;
        }

        getEspelmes () {
```


Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

```
            return this.espelmes;
        }

        setEspelmes (v) {
            this.espelmes = v;
        }

        aTexto () {
        return ` Soy un pastel con el nombre ${super.getNom()} de sabor de
                  ${super.getSabor()} con un peso de ${super.getPes()} Kg, de color
                            ${super.getColor()} para ${super.getRacions()} raciones
                                                    con ${this.espelmes} velas`;
        }
    }

    const article1 = new Brioixeria(
        'Croissant', 'mantequilla', 0.15, 'beige', 1
    );

    document.writeln( "<p>" + article1.aTexto() + "</p>");

    const article2 = new Pastis(
        'Pastel de manzana', ' manzana y caramelo', 1.5, 'marrón', 6, 12
    );

    document.writeln( "<p>" + article2.aTexto() + "</p>");
</script>
```


Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

JQUERY

jQuery es una librería de JavaScript creada para simplificar tareas habituales en el desarrollo web. Nació con el objetivo de hacer el código más corto, más legible y más compatible entre navegadores en una época en la que cada navegador interpretaba JavaScript de manera diferente. Su filosofía se resume en el eslogan "*Write less, do more*" –Escribe menos, haz más-.

Con jQuery puedes seleccionar elementos del DOM con una sintaxis muy compacta, manipular contenido HTML, manipular clases y estilos CSS, gestionar eventos, crear animaciones y hacer peticiones AJAX sin tener que preocuparte por los detalles técnicos de cada navegador. Su función principal, **$()**, actúa como una puerta de entrada rápida para acceder y modificar elementos de la página.

Aunque hoy en día JavaScript moderno (ES6+) y la API del DOM han mejorado mucho y han reducido la necesidad de jQuery, todavía se encuentra en muchos proyectos existentes y es útil para mantener o modernizar aplicaciones antiguas. También sigue siendo una herramienta sencilla para quien empieza y quiere manipular el DOM de manera intuitiva.

En el capítulo dedicado al DOM, hemos visto algunos métodos y propiedades (no todos) de JavaScript para manipularlo, y hemos constatado que hay una mezcla de propiedades, métodos y métodos de propiedades. jQuery simplifica la sintaxis definiendo todo como métodos y variando sólo los argumentos de cada uno.

Además, gracias al amplio catálogo de ***plugins*** o extensiones desarrolladas con esta librería, es muy fácil implementar nuevas funcionalidades o conectarlas entre ellas sin que estudiar su sintaxis o cómo hacerlas compatibles entre ellas.

jQuery no sustituye al JavaScript: lo complementa para facilitar la manipulación de DOM con una sintaxis compacta basada en métodos que resumen largas expresiones.

INSTALACIÓN

Al ser una librería como Bootstrap o FontAwesome, no es necesario instalar ningún software especial: solo hay que enlazar con el archivo base que nos interese, disponibles en https://jquery.com/download/

- Archivo minificado con todas las funcionalidades: https://code.jquery.com/jquery-3.7.1.min.js
- Archivo *sourcemap* -fichero que sirven para relacionar el código minificado con el código original legible y que se pueda leer con claridad el código en las herramientas de depuración del navegador- complementario al minificado: https://code.jquery.com/jquery-3.7.1.min.map
- Archivo sin minificar –no recomendado en producción por su peso- para el desarrollo: https://code.jquery.com/jquery-3.7.1.js
- Archivo minificado simplificado –slim- sin efectos visuales ni ajax: https://code.jquery.com/jquery-3.7.1.slim.min.js
- Archivo sourcemap de la versión simplificada: https://code.jquery.com/jquery-3.7.1.slim.min.map
- Archivo sin minificar simplificado: https://code.jquery.com/jquery-3.7.1.slim.js

Este archivo normalmente lo descargaremos y lo guardaremos en la carpeta del proyecto, con el resto de recursos. También tenemos la posibilidad de utilizar un **CDN** (*content delivery network*) o red de entrega de contenidos donde hay copias disponibles de los archivos.

Una vez descargado o con el enlace CDN que nos interese, lo vincularemos con una etiqueta `<script>` al `<head>` de nuestro documento, aunque normalmente se pone al final del `<body>`, con el resto de scripts, para no enlentecer la carga del documento:

```
<script src="jquery-3.7.1.min.js"></script>
```

Es importante que enlacemos a la librería jQuery antes del resto de scripts que utilicen su sintaxis para evitar errores de lectura.

INICIALIZACIÓN

Si jQuery está pensado para manipular el DOM, antes de empezar a utilizarlo debemos asegurarnos de que todo el documento esté completamente cargado, por lo tanto, usaremos una expresión similar al `window.addEventListener("load",function(){});` para esperar a ejecutar el código al que los nodos estén disponibles:

```
<script src="jquery-3.7.1.min.js"></script>
<script>
    $(document).ready(function(){
        ...
    });
</script>
```

Aunque hay una versión más cómoda y rápida:

```
<script src="jquery-3.7.1.min.js"></script>
<script>
    $(function(){
        ...
    });
</script>
```

A partir de ahora, todas nuestras sentencias irán dentro de las llaves de esta función anónima que se invocará una vez el documento esté cargado.

SELECTORES

A partir de ahora, cada vez que implementamos una nueva sentencia, la estructura será clara: **selector.método()**;

Para hacer el selector necesitamos la expresión propia del jQuery **`$("selector")`**, donde el selector puede ser cualquiera de los posibles selectores que nos da el CSS. Esta expresión es similar al:
`document.querySelectorAll("selector");`

Aparte de simplificar, jQuery también hará que cualquier método que apliquemos al selector se aplique a todos los elementos, es decir, el mismo jQuery hará el bucle `forEach()` por nosotros:

```
<script src="jquery-3.7.1.min.js"></script>
<script>
    $(function(){
        $('.oculto').hide();
    });
</script>
```


Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

MÉTODOS

El listado de métodos que muestra la documentación de jQuery en https://api.jquery.com/ es muy amplio, por eso recomiendo otra webapp que he desarrollado como guía visual, donde los métodos aparecen organizados por temática: https://onaweb.cat/jquery/

Al pulsar cada uno de los métodos, se abre una ventana con la documentación original en inglés, donde no sólo se explican los argumentos necesarios, sino también hay ejemplos de uso.

Cabe mencionar que el JavaScript tiene muchas propiedades que pueden ser de lectura o escritura si están a la derecha o a la izquierda del operador de asignación. En jQuery también están definidos como métodos, pero estos serán de lectura –*getters*- si sólo tienen un argumento, o de escritura –*setters*- si tienen dos argumentos.

Si alguno de los argumentos es de tipo función, como los *handler* de los métodos de eventos, será una función anónima donde podremos añadir el código a ejecutar en el caso de que el método se ejecute. Dentro de estas funciones anónimas utilizaremos la cláusula **this** para hacer referencia al objeto con el que se está interactuando en ese momento, así que el uso de la función anónima o función de flecha podría dar errores en la interpretación del **this**.

Otra opción que tiene el JavaScript y también el jQuery es la opción de concatenar diferentes métodos con el punto: **`$("selector").método1().método2().método3()...;`**

Algunos de estos métodos tienen la cláusula **`jQuery`** o **`$`** en lugar de un selector porque no es necesario aplicar a un nodo en concreto: se pueden emplear como una función.

En el siguiente ejemplo hay tres botones con la misma clase CSS pero diferentes textos dentro, y un contenedor identificado vacío. A continuación está el enlace a la liberia jQuery y su inicialización, donde hay un selector para los botones por su clase y un método de evento con dos argumentos: el evento "click" y la función anónima con dos sentencias:

```
<p>
    <button type="button" class="btnBoton">Botón 1</button>
    <button type="button" class="btnBoton">Botón 2</button>
    <button type="button" class="btnBoton">Botón 3</button>
</p>
<div id="salida"></div>

<script src="jquery-3.7.1.min.js"></script>
<script>
    $(function(){
        $('.btnBoton').on('click',function(){
            let textoBoton = $(this).text();
            $('#salida').text(textoBoton)
        });
    });
</script>
```

En la primera sentencia se declara la variable local "textoBoton" y se instancia con el selector **`$(this)`** que equivale al botón exacto sobre el que se ha pulsado con el método **`text()`** que equivale a la propiedad `innerText` o `textContent`; al no tener ningún argumento, este método funciona como lectura –*getter*-.

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

En la siguiente sentencia volvemos a empezar con un selector, en este caso el contenedor con el identificador, y volvemos a aplicar el método **`text()`** que, en esta ocasión, sí tiene un argumento, por tanto, funciona como escritura –*setter*- y muestra el contenido de la variable "textoBoton".

SELECTORES

En esta categoría tenemos un resumen de las diferentes formas que tenemos para seleccionar nodos del DOM, la mayoría heredados del lenguaje CSS.

ATRIBUTOS / CSS

En esta categoría tenemos los métodos para manipular (leer o escribir) los atributos de las etiquetas HTML y las propiedades CSS, así como las clases CSS que se aplican. También se incluyen métodos para manipular las dimensiones, la posición en la ventana y los atributos `data-` de las etiquetas HTML.

Este atributo "data" merece una subcategoría aparte porque es una forma muy habitual de almacenar información propia para esta etiqueta (como una variable local) para luego recuperarla con JavaScript y, en nuestro caso, con jQuery.

Siguiendo el ejemplo inicial del capítulo, se han añadido atributos `data-info` a los botones para poder personalizar el texto a mostrar. Por lo tanto, cada vez que pulsamos un botón debemos recuperar la información de ese mismo botón, y lo hacemos con el método **`data('info')`** donde "info" es el argumento para especificar cuál de todos los atributos `data-` que puede tener una etiqueta queremos seleccionar. Como no hay un segundo argumento de valor, significa que es de lectura –*getter*-:

```
<p>
    <button type="button" class="btnBoton" data-info="Información 1">Botón 1</button>
    <button type="button" class="btnBoton" data-info="Información 2">Botón 2</button>
    <button type="button" class="btnBoton" data-info="Información 3">Botón 3</button>
</p>
<div id="salida"></div>
<script src=" jquery-3.7.1.min.js "></script>
<script>
    $(function(){
        $('.btnBoton').on('click',function(){
            let textoBoton = $(this).data('info');
            $('#salida').text(textoBoton)
        });
    });
</script>
```

MANIPULACIÓN

Son métodos para manipular el contenido o lo que lo rodea, duplicar o incluso eliminar el nodo seleccionado. En el ejemplo anterior se ve el uso del método **`text()`** para manipular el texto contenido en un nodo.

ATRAVESANDO

Este conjunto de métodos nos permite movernos por la estructura de nodos del documento. Siempre necesitamos empezar desde un nodo origen, el seleccionado, y a partir de estos nos podemos mover a nodos inferiores –los hijos-, superiores –los padres- o los que están al mismo nivel –los hermanos-.

EVENTOS

Este conjunto de métodos nos permiten detectar las interacciones del usuario con el ratón, el teclado o relacionados con los formularios.

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

En el siguiente ejemplo se modifica un ejemplo anterior para utilizar la sintaxis jQuery: en este caso el selector del formulario (o formularios) ahora es más sencillo, no es necesario utilizar ningún bucle pero si queremos seguir utilizando métodos originales del JavaScript -`.checkValidity()` y `.submit()`- y que el jQuery no se confunda, añadimos el método **`.get(0)`** para indicar que en el array de posibilidades del jQuery –para jQuery todo conjunto de elementos es un array, aunque sea de un único elemento-, nos quedamos con el primer nodo –el formulario que estamos manipulando-:

```
<form class="para-validar" novalidate>
    <p>
        <label for="idNombre">Nombre</label>
        <input type="text" id="idNombre" required name="elNombre">
    </p>
    <p>
        <button type="submit">Enviar</button>
    </p>
</form>
<script src=" jquery-3.7.1.min.js "></script>
<script>
    $(function(){
        $('.para-validar').on("submit", function(event) {
            event.preventDefault();
            event.stopPropagation();
            if ( $(this).get( 0 ).checkValidity() ) {
                $(this).get( 0 ).submit();
            } else {
                alert("Verifica los campos del formulario");
            }
        });
    });
</script>
```

EFECTOS

Aquí tenemos métodos para efectos visuales; no son efectos muy complejos, pero para hacer efectos sencillos para mejorar la experiencia de usuario pueden ser un buen punto de partida.

Siguiendo el ejemplo anterior, no emplearemos un `alert()` sino un mensaje en pantalla ya definido en el código HTML:

```
<form class="para-validar" novalidate>
    <p>
        <label for="idNombre">Nombre</label>
        <input type="text" id="idNombre" required name="elNombre">
    </p>
    <p>
        <button type="submit">Enviar</button>
    </p>
</form>
<div class="aviso">Verifica los campos del formulario</div>
<script src=" jquery-3.7.1.min.js "></script>
<script>
    $(function(){
        $('.aviso').hide();
        $('.para-validar').on("submit", function(event) {
            event.preventDefault();
            event.stopPropagation();
            if ( $(this).get( 0 ).checkValidity() ) {
                $(this).get( 0 ).submit();
            } else {
```


Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

```
                $(this).next('.aviso').slideDown();
            }
        });
    });
</script>
```

En este ejemplo, al cargar el documento ocultamos el nodo con el mensaje, y cuando se valida y da un error de validación, desde el formulario nos movemos al siguiente nodo (el del mensaje) para desplegarlo.

AJAX

AJAX es una técnica de JavaScript que permite comunicarse con el servidor sin recargar toda la página. El nombre proviene de ***Asynchronous JavaScript and XML***, aunque hoy en día no es necesario utilizar XML: también funciona con JSON, texto o cualquier otro formato.

La asincronía es la capacidad de ejecutar tareas sin bloquear la aplicación, permitiendo que otras operaciones continúen mientras una acción lenta se resuelve en segundo plano.

La idea central es sencilla: la página envía una petición al servidor "en segundo plano", recibe la respuesta y actualiza sólo la parte necesaria del documento. Esto hace que las aplicaciones web sean más rápidas, fluidas e interactivas, porque el usuario no ve ningún corte ni recarga completa.

Con AJAX puedes, por ejemplo, cargar datos nuevos, enviar formularios, actualizar listas o validar información sin salir de la página. Inicialmente se usaba el objeto `XMLHttpRequest`, pero hoy es muy habitual utilizar la API `fetch`, que es más moderna y clara.

En resumen, AJAX es el mecanismo que permite que muchas webs funcionen de manera dinámica y reactiva, haciendo que la comunicación con el servidor sea transparente para el usuario.

jQuery incluye su implementación con el método **`ajax()`** y el resto de métodos auxiliares para facilitar la gestión de datos y respuestas.

```
<div id="galeria"></div>
<script src=" jquery-3.7.1.min.js "></script>
<script>
    $(function(){
        $.getJSON( "https://picsum.photos/v2/list", function( data ) {
            let items = [];
            $.each( data, function( key, val ) {
                items.push( `<div><img src="${val.download_url}" alt="foto
                                        ${val.id}" title="${val.author}"></div>` );
            });
            $("#galeria").append( items.join('') );
        });
    });
</script>
```

En el anterior ejemplo se ve en uso el método **`$.getJSON()`** para conectar con un servidor que da un documento .json con una lista de 30 imágenes; una vez establecida la conexión, leído el archivo y descodificado en formato array, lo podemos recorrer para almacenar en el array local "items" los nuevos nodos de HTML con imágenes. Los valores de los atributos de las imágenes se establecen a partir de la información del **JSON** (***JavaScript Object Notation***). Una vez recorrido, se muestra el contenido dentro del contenedor con el identificador "galeria".

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

NÚCLEAR

Aquí tenemos un "cajón desastre" donde tenemos el resto de métodos que no pertenecen al resto de categorías. Muchos métodos son ayudas y auxiliares, como el método `$.each()` –que hemos visto en el ejemplo anterior- para hacer un bucle a los elementos de un array o el método `.get()` –que ya hemos visto en ejemplos anteriores- para seleccionar el nodo nativo de JavaScript de un array de nodos del jQuery.

PLUGINS

Un *plugin* de JavaScript es un pequeño módulo de código pensado para añadir funcionalidades concretas a una página o aplicación sin tener que reescribirlas desde cero. Actúa como una extensión reutilizable: encapsula una característica (por ejemplo, un carrusel, un selector de fechas o un sistema de pestañas) y permite integrarla fácilmente en diferentes proyectos mediante una API sencilla.

Pero depende del autor cómo están desarrollados y la sintaxis utilizada, y eso implica que hay que leer muy bien la documentación y estudiar su funcionamiento para entender cómo personalizarlo y hacerlo funcionar con el resto del código del proyecto.

En el ecosistema de jQuery, por ejemplo, los *plugins* han sido una manera muy popular de compartir soluciones y ampliar las capacidades de la librería y, además, al estar escritos todos bajo las mismas premisas, son fáciles de entender y de implementar.

Cada vez que queramos añadir un nuevo *plugin*, habrá que seguir las siguientes **reglas de oro**:

1. **Encontrar la página web de documentación del *plugin***: a veces el autor del *plugin* ha creado un sitio web específico para documentar el *plugin*, pero normalmente los encontraremos en la plataforma de github.com.
2. **Descargar los archivos del *plugin***: en la misma web del autor encontraremos el enlace de descarga o, si ya estamos en GitHub, podemos descargarnos el lote entero.
3. **Localizar los archivos necesarios para copiarlos en nuestro proyecto**: leyendo la documentación veremos qué .css y .js necesitaremos para hacer funcionar el *plugin*. Estos archivos serán los necesarios que copiemos en la carpeta del proyecto. Normalmente, los encontramos en la carpeta `/dist/`.
4. **Enlazar los archivos en el código HTML**: es importante que el enlace a los archivos .js se hagan después del enlace a la librería de jQuery, pero antes de nuestro documento .js donde añadimos nuestro código. Es importante aclarar que no trabajaremos en los archivos del *plugin* de la misma forma que no trabajamos en el archivo del jQuery.
5. **Crear la estructura HTML**: los *plugins* tienen una aplicación práctica sobre un contenido de nuestro documento y debemos crearlo para ver sus resultados.
6. **Inicializar el *plugin***: los *plugins* escritos por jQuery están definidos como nuevos métodos, por lo tanto, hay que leer la documentación para ver qué nombre tienen y ver qué argumentos u opciones de configuración tienen.

OWLCAROUSEL

Para mostrar cómo podemos emplear un *plugin*, explicaremos el *plugin* de OwlCarousel para jQuery, una librería para poder hacer pases de diapositivas o carruseles dinámicos.

1. **Página web de documentación**: https://owlcarousel2.github.io/OwlCarousel2/
2. **Descargar los archivos del *plugin*:** https://github.com/OwlCarousel2/OwlCarousel2/archive/2.3.4.zip

Generalitat de Catalunya
MINISTERIO DE EDUCACIÓN, FORMACIÓN PROFESIONAL Y DEPORTES
MINISTERIO DE TRABAJO Y ECONOMÍA SOCIAL
SERVICIO PÚBLICO DE EMPLEO ESTATAL SEPE SERVEI PÚBLIC D'OCUPACIÓ ESTATAL
Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

3. **Localizar los archivos necesarios:** al descomprimir encontramos una carpeta `/dist/` con los archivos `owl.carousel.js` y `owl.carousel.min.js`. Como el segundo es la versión minificada del primero, más ligera, es la que despegaremos en nuestro proyecto. Además, según la documentación, de la carpeta `/dist/assets/` necesitaremos el archivo `owl.carousel.min.css` por los estilos básicos y `owl.theme.default.min.css` o `owl.theme.green.min.css` para añadir los controladores predeterminados.
4. **Enlazar los archivos:** en nuestro ejemplo, quedaría algo parecido a:

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Carrusel con OwlCarousel</title>
    <link rel="stylesheet" href="OwlCarousel2-2.3.4/dist/assets/owl.carousel.min.css">
    <link rel="stylesheet" href="OwlCarousel2-2.3.4/dist/assets/owl.theme.default.min.css">
    <style>
        body { font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;}
    </style>
</head>
<body>
    <h1>Carrusel amb OwlCarousel</h1>

    <script src="jquery-3.7.1.min.js"></script>
    <script src="OwlCarousel2-2.3.4/dist/owl.carousel.min.js"></script>
    <script src="funcions.js"></script>
</body>
</html>
```

5. **Crear la estructura HTML**: creamos un contenedor con una lista de imágenes para mostrarlas en carrusel:

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Carrusel con OwlCarousel</title>
    <link rel="stylesheet" href="OwlCarousel2-2.3.4/dist/assets/owl.carousel.min.css">
    <link rel="stylesheet" href="OwlCarousel2-2.3.4/dist/assets/owl.theme.default.min.css">
    <style>
        body { font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;}
    </style>
</head>
<body>
    <h1>Carrusel amb OwlCarousel</h1>
    <div id="galeria">
        <img src="https://picsum.photos/id/0/960/540" alt="foto 0" title="foto 0">
        <img src="https://picsum.photos/id/1/960/540" alt="foto 1" title="foto 1">
        <img src="https://picsum.photos/id/2/960/540" alt="foto 2" title="foto 2">
        <img src="https://picsum.photos/id/3/960/540" alt="foto 3" title="foto 3">
        <img src="https://picsum.photos/id/4/960/540" alt="foto 4" title="foto 4">
    </div>
    <script src="jquery-3.7.1.min.js"></script>
    <script src="OwlCarousel2-2.3.4/dist/owl.carousel.min.js"></script>
    <script src="funcions.js"></script>
</body>
</html>
```


Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)

6. **Inicializar el *plugin***: en esta librería debemos hacer dos cosas; primero, añadir las clases css al contenedor de galería:

```
    <div id="galeria" class="owl-carousel owl-theme">
        ...
    </div>
```

Segundo, hacer el selector del contenedor y aplicar el método del *plugin* en nuestro archivo funcions.js:

```
$(function(){
    $('#galeria').owlCarousel({
        loop:true,
        margin:10,
        nav:true,
        responsive:{
            0:{
                items:1
            },
            600:{
                items:3
            },
            1000:{
                items:5
            }
        }
    });
});
```

A partir de ahora, depende del autor de qué argumentos podemos añadir al método para configurar su funcionamiento. Todas estas opciones estarán disponibles en la documentación de la web.

Aquesta actuació està impulsada i subvencionada pel Servei Públic d'Ocupació de Catalunya (SOC) amb fons rebuts del Ministeri d'Educació, Formació Professional i Esport i del Servei Públic d'Ocupació Estatal (SEPE)