Javascript

INTRODUCCION JAVASCRIPT

Javascript es un lenguaje de programación creado por la empresa **Netscape en 1995**. Es un **lenguaje de programación interpretado** a través de un navegador en el lado del cliente. Un lenguaje interpretado significa que las instrucciones las analiza y las procesa un navegador en el momento de ser ejecutadas.

Para aprender Javascript es imprescindible conocer previamente los otros dos "lenguajes" (no se consideran como tal aunque se utiliza de programación del lado cliente que conforman las capas para el desarrollo de páginas web:

- **HTML**: Para definir la capa de contenido y estructura de las páginas web.
- **CSS**: Para definir la capa de diseño o presentación de las páginas web.
- **Javascript**: Lenguaje para programar el comportamiento de las páginas web desde el cliente.

Javascript NO es Java, aunque el nombre sea muy similar. Se consideran lenguajes totalmente diferentes. Todos los lenguajes de programación comparten una serie de conceptos básicos como:

- Lenguaje de programación: Son una serie de instrucciones que interpreta el ordenador para realizar determinadas tareas. Cada lenguaje de **programación tiene su propia sintaxis**, su estructura y su forma de ser ejecutado. [IDIOMA].

- Algoritmo: A través del lenguaje de programación creamos algoritmos o rutinas que son los **pasos a seguir para resolver un problema**. Un algoritmo puede estar compuesto por varios algoritmos sin necesidad de ser un programa. [FRASE].

- Programa: Son un conjunto de instrucciones organizadas con la finalidad de resolver problemas en las aplicaciones. Cada problema se resuelve mediante su correspondiente algoritmo. Un **programa contiene miles de algoritmos interrelacionados** entre sí. [LIBRO]

Javascript fue creado por **Brendan Eich** en 1995 y se convirtió en un estándar en 1997. Es un lenguaje que se encuentra en constante evolución. Actualmente **ECMA-262** es el nombre oficial de la norma y ECMAScript es el nombre oficial del lenguaje. La versión **ECMAScript 5** (Junio 2011) es la más aceptada y asimilada por todos los navegadores. En la actualidad existen otras evoluciones como **ECMAScript 6** (Junio 2015) y **ECMAScript 7** (en desarrollo) que no son 100% compatibles con todos los navegadores modernos. La última versión publicada es de Junio 2019 y por convenio se actualiza anualmente.

Ed.	Fecha	Nombre formal / informal	Cambios significativos
1	JUN/1997	ECMAScript 1997 (ES1)	Primera edición
2	JUN/1998	ECMAScript 1998 (ES2)	Cambios leves
3	DIC/1999	ECMAScript 1999 (ES3)	RegExp, try/catch, etc...
4	AGO/2008	ECMAScript 2008 (ES4)	Versión abandonada.
5	DIC/2009	ECMAScript 2009 (ES5)	Strict mode, JSON, etc...
5.1	DIC/2011	ECMAScript 2011 (ES5.1)	Cambios leves

A partir del año 2015, se marcó un antes y un después en el mundo de Javascript, estableciendo una serie de cambios que lo transformarían en un lenguaje moderno, partiendo desde la especificación de dicho año, hasta la actualidad:

Ed.	Fecha	Nombre formal / informal	Cambios significativos
6	JUN/2015	ECMAScript 2015 (ES6)	Clases, módulos, generadores, hashmaps, sets, for of, proxies...
7	JUN/2016	ECMAScript 2016	Array includes(), Exponenciación **
8	JUN/2017	ECMAScript 2017	Async/await
9	JUN/2018	ECMAScript 2018	Rest/Spread operator, Promise.finally()...
10	JUN/2019	ECMAScript 2019	Flat functions, trimStart(), errores opcionales en catch...
11	JUN/2020	ECMAScript 2020	Dynamic imports, BigInt, Promise.allSettled

Características importantes de JavaScript:

- **Javascript NO es Java**, aunque la similitud de los nombres genera muchas confusiones.

  - Java es un lenguaje de programación de bajo nivel (como C, C++, Delphi) y de propósito general, es decir, se pueden crear programas en Java para ser ejecutados en servidores, en equipos locales de sobremesa, en Tablets, en Smartphones o en navegadores (Applets). Java es un **lenguaje compilado**, es decir, se transforma a código máquina (binario) y empaquetado en un archivo ejecutable previo a su ejecución. Con Java se crean **aplicaciones nativas** específicas para cada uno de los sistemas operativos.

  - Javascript es un lenguaje de programación que **se ejecuta exclusivamente en un navegador web** (host), **multiplataforma**, **orientado a objeto** y ha sido diseñado únicamente para que se ejecute en el navegador. JS es un **lenguaje interpretado**, es decir, se compila cuando se ejecuta por medio de un programa que reside en la máquina donde se ejecuta el código fuente. El programa es el navegador web. Con Javascript se pueden crear **aplicaciones web y aplicaciones híbridas**.

- JS fue diseñado para **añadir interactividad y efectos visuales a las páginas web**. La estructura de una web la aporta HTML y el diseño el CSS. El estándar recomienda tenerlo todo bien separado.

- JS es un **lenguaje del lado cliente** para el diseño de páginas y aplicaciones web en internet y se ejecuta en el equipo local ganando en velocidad de respuesta.

- JS es un lenguaje **interpretado embebido** (incrustado) en el código de una página web HTML. Dicho de otra forma JS requiere de un intérprete en el navegador para poder ser ejecutado. Además no todos los navegadores utilizan el mismo intérprete por lo que puede haber pequeñas diferencias entre unos y otros.

- JS es un lenguaje de secuencias de comandos que puede **reaccionar a eventos**. Un evento es una acción como, por ejemplo, un clic de ratón, un movimiento de ratón, pulsar una tecla, un giro sobre una tableta, etc…

- JS puede **leer y modificar el contenido de un elemento HTML**.

- JS se utiliza para **la validación de contenido en los formularios** antes de ser enviados al servidor. Se consigue acelerar los tiempos de respuesta y no cargar al servidor con más tareas. Todo y con eso el estándar HTML5 de formulario ya realiza validaciones de forma automática.

- JavaScript **puede utilizarse para crear cookies**. Una cookie se utiliza para almacenar y recuperar información del usuario cuando está navegando. La finalidad de la cookie es acelerar el proceso de carga de la web como por ejemplo al rellenar la información de un formulario que previamente ya ha sido rellenado.

- Cualquier persona puede utilizar JavaScript **sin necesidad de adquirir una licencia**. Es un lenguaje gratuito y por eso se ha expandido tanto en los últimos años. Además, la tendencia actual es generar aplicaciones web universales y JS facilita la creación de software ya que permite la visualización multiplataforma.

Software de ejemplo (editores de pago y gratuito) para programar en JavaScript

De pago (profesionales)	Gratuitos
Dreamweaver CS5, CS6, CC.	Sublime Text (*)
Microsoft FrontPage.	Visual Studio Code (*)
Adobe PageMill.	Atom (*)
CutePage.	NetBeans
EditPlus	Notepad ++

* son los editores que "mayoritariamente" utilizan los programadores web.

Algunos navegadores actuales han sido diseñados para facilitar la tarea de creación a los desarrolladores web. Por este motivo los grandes navegadores tienen diferentes versiones para el desarrollo web como Google o Firefox que disponen de versiones específicas de sus navegadores para los desarrolladores como **Google Chrome Canary** y **Firefox Developer** respectivamente.

Estas versiones adaptadas tienen sus pros y contras.

- **Pros**: Se actualizan a diario y tienen funcionalidades que no están en las versiones más habituales. **Se pueden probar las nuevas características de los navegadores** sobre nuestras aplicaciones. Además, está disponible para un menor número de sistemas operativos.

- **Contras**: Son inestables y **pueden dejar de funcionar sin motivo aparente**. Las actualizaciones las añaden y las quitan sin previo aviso. El resultado de la aplicación puede no visualizarse como uno desearía debido a estas actualizaciones.

PARADIGMAS DE LA PROGRAMACION

Un paradigma en programación es un estilo de desarrollo de programas, es decir, **un modelo para resolver problemas computacionales**. Los lenguajes de programación, necesariamente, se encuadran en uno o varios paradigmas a la vez a partir del tipo de órdenes que permiten implementar, algo que tiene una relación directa con su sintaxis.

- **Programación Imperativa**. Los programas se componen de un conjunto de **sentencias una detrás de otra que cambian su estado**. Son secuencias de comandos que ordenan acciones a la computadora de forma secuencial, es decir una tras otra. El famoso GOTO que generaba 'código espagueti' surgió con este tipo de programación. A raíz de esta forma de programar se han desarrollado nuevos sistemas:

  - **Programación Estructurada**: El programador se centra en el algoritmo para la resolución de un problema. En este paradigma **se trabajan con funciones** que reciben recursos (parámetros) y devolviendo valores después de ser procesados.

  - **Programación Modular**: Surge a raíz que cada vez los programas son más extensos y el código se convierte ilegible debido a la gran cantidad de líneas de código. La programación estructurada no consigue gestionar tanto volumen de información. **Los programas están divididos en módulos independientes**, cada uno con un comportamiento bien definido, que se pueden reutilizar en otras aplicaciones.

  - **Programación Orientado a Objetos**. El comportamiento del programa es llevado a cabo por objetos, entidades que representan elementos del problema a resolver y tienen atributos y comportamientos. Los objetos son fácilmente exportables y reutilizables por otros desarrolladores.

  - **Programación dirigida a Eventos**. El flujo del programa está determinado por sucesos externos generados por los usuarios. Por ejemplo, la acción que realiza un usuario cuando hace clic con el mouse, al entrar en un elemento tipo <input>, o al posicionarnos sobre un elemento de la web). Javascript es un lenguaje especialmente diseñado para trabajar con eventos.

- **Programación Declarativa**. Opuesto al imperativo. Los programas describen los resultados esperados sin listar explícitamente los pasos a llevar a cabo para alcanzarlos.
    - **Lógico**. El problema se modela con enunciados de lógica de primer orden.
    - **Funcional**. Los programas se componen de funciones, es decir, implementaciones de comportamiento que reciben un conjunto de datos de entrada y devuelven un valor de salida.

Javascript es uno de los muchos lenguajes de programación que **acepta trabajar con uno o varios paradigmas a la vez**, es más, a la hora de programar se utilizan varios paradigmas.

No hay un paradigma mejor que otro, pero sí es probable que **un paradigma puede ser más adecuado para solucionar un tipo de problema que otro**. El programador a base de experiencia podrá decidir en cada momento cual es el paradigma que más le conviene en cada caso.

Cabe destacar que tanto en lenguajes de alto nivel o bajo nivel el paradigma que utilizan por excelencia debido a las propias características del mismo es el **paradigma orientado a objetos**. Hoy en día más del 95% del software que se crea utiliza el paradigma de la POO en combinación con otros paradigmas como la estructura o modular.

Todos los programadores de aplicaciones **antes de empezar a programar en POO es necesario que dominen otros paradigmas**. La POO es una forma de solucionar problemas más eficientes, pero requiere de un óptimo nivel de programación por parte del programador y un conocimiento exhaustivo del lenguaje y su funcionamiento.

SINTAXIS BÀSICA JAVASCRIPT

Añadiendo código JS dentro de etiquetas **<head>** o **<body>**

En HTML un código JavaScript debe ir encerrado entre las etiquetas **<script>** y **</script>**. Es la forma de indicar que empieza código JavaScript. El código JavaScript puede ir dentro de las etiquetas **<head>** o **<body>**, aunque lo más habitual es acceder a él a través de un **enlace externo al archivo** que contiene los procedimientos desde el documento HTML igual que si fuese un archivo CSS.

En los siguientes ejemplos se muestra como el código Javascript puede estar insertado en diferentes zonas del archivo principal .html.

Ejemplo 1 (<head>): En este primer caso el código es leído por el navegador y se ejecuta con la misma carga de la página. Si se opta por este sistema se debe introducir el código JS entre las etiquetas **<script>** para diferenciarlo de cualquier otro tipo de código. Este sistema aunque se sigue utilizado **NO es el más recomendable** según el estándar HTML5 porque no separa los tres elementos principales de una página web (Estructura – Diseño – Interactividad).

```
// El código JS se introduce entre las etiquetas <head>
<html>
  <head>
     <script>
          document.getElementById("caja1").innerHTML ="JavaScript";
     </script>
  </head>
  <body>
     Contenido página web
  </body>
</html>
```

Ejemplo 2 (<body>): Este sistema de **código embebido** en el **<body>** también se ejecuta en el momento que la página se carga. Una de las características más interesantes de Javascript es poder ejecutar código después de haber realizado la carga de la web. Por tanto, aunque este sistema sigue vigente y está permitido el estándar HTML5 no recomiendo su uso porque no separa los tres elementos principales de una página web (Estructura – Diseño – Interactividad).

```
// Código introducido entre las etiquetas <body>

<html>
  <head>
  </head>
  <body>
     <script>
          document.write('Hola a todos, soy yo mismo');
          document.write('<br>'); // Elemento HTML intercalado
          document.write('Programación en JavaScript');
     </script>
  </body>
</html>
```

Ejemplo 3 (llamada a función): Las funciones en JS se pueden declarar en el **<head>** de la web para posteriormente ser invocadas desde nuestro código utilizando el sistema de eventos En este caso se crea una función que se invocará desde la etiqueta **<body>** mediante un evento. Las funciones y rutinas se suelen incluir en un archivo externo JavaScript con la extensión .js pero también es posible introducirlas dentro de la propia página.

```
// Función JS declarada el <head> se invoca desde HTML con un evento
// La llamada se realiza desde el botón pulsar que está en el <body>
<html>
  <head>
     <script>
          function miFuncion() {
               document.getElementById("caja1").innerHTML = "Cambia
               texto";}
     </script>
  </head>
  <body>
     <button type="button" onclick="miFuncion()"> Pulsar </button>
  </body>
</html>
```

JavaScript en **archivo externo** con extensión .js

La forma más habitual de enlazar el documento HTML con procedimientos en JS es **mediante hojas externas**, de la misma forma que se realiza con CSS. El enlace con el archivo externo de JavaScript se realiza siempre dentro de las etiquetas **<head>**. Las ventajas de utilizar hojas externas son:

- Mantiene separado la estructura HTML del código.
- Hace más fácil la lectura y el mantenimiento tanto del HTML como de JavaScript.
- Al estar cargados los archivos .js en caché aceleran notablemente la carga de la página.

Ejemplo 1: Enlace **especificando la URL** completa donde se encuentra el archivo de funciones.

```
<script src="https://www.miweb.com/js/myScript1.js"> </script>
```

Ejemplo 2: Enlace **especificando una ruta** desde el sitio actual. Es una referencia relativa. Los caracteres punto punto barra (..) se utilizan para subir un nivel en la estructura de árbol de las carpetas.

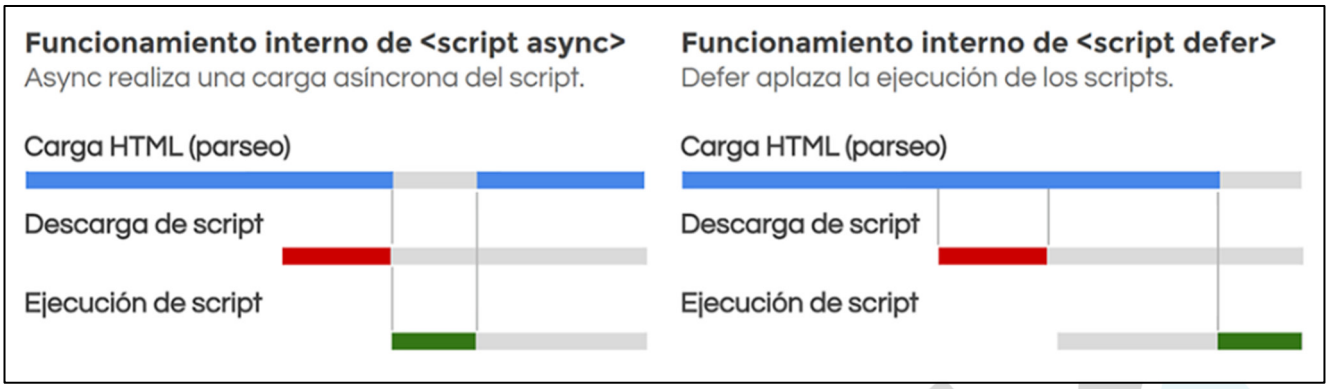
```
<script src="../js/myScript1.js"></script>
```

Ejemplo 3: Enlace de 2 archivos .js se cargan de diferente forma (async, defer)

```
<head>
   <script src="file_js1.js" async> </script>
   <script src="file_js2.js" defer> </script>
</head>
```

async: La página se carga de forma asíncrona, es decir, el código se ejecuta mientras se procesa el resto del documento.

defer: El código .js se ejecuta después de que la página se haya cargado por completo. Solo funciona con scripts externos al HTML.



Es posible también enlazar desde las etiquetas **<head>** con las **librerías de Javascript**, como por ejemplo JQuery. Una librería es un archivo o conjunto de archivos (también llamado "*framework*") que se utiliza para facilitar la programación de un website. De esta forma es posible reutilizar código que ya está creado y que funciona correctamente en todos los navegadores.

Técnicamente una librería y un framework no son lo mismo ya que el segundo es un entorno de trabajo mientras que el primero es un conjunto de métodos agrupados bajo el nombre de librería. Para nuestro objetivo por ahora podemos entenderlos como un recurso externo al standard de Javascript.

```
<script src=" Jquery/jquery-1.6.3.min.js"> </script>
```

ALERTAS Y VENTANAS DE DATOS

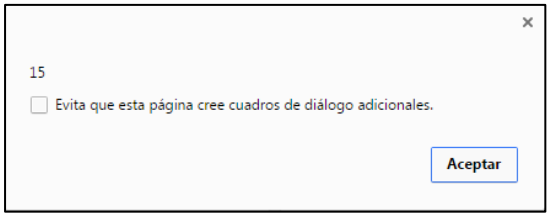
En JavaScript existen **diferentes formas de enviar o recibir información** al equipo, y de modificar los datos que se muestran de la página web. Los más significativos y utilizados se muestran a continuación:

1. **document.write**: Añade texto en el **<body>** de una página. Borra el contenido de la web (si lo hubiera) y escribe el contenido pasado a través de los parámetros. Es utilizado para testear las aplicaciones, aunque para programadores es más útil 'console.log'. Es un sistema poco utilizado en la actualidad pero para empezar a conocer el lenguaje tiene su utilidad.

```
// Escribe el número 15 en el navegador directamente

<script>

      document.write(5 + 10);

</script>
```

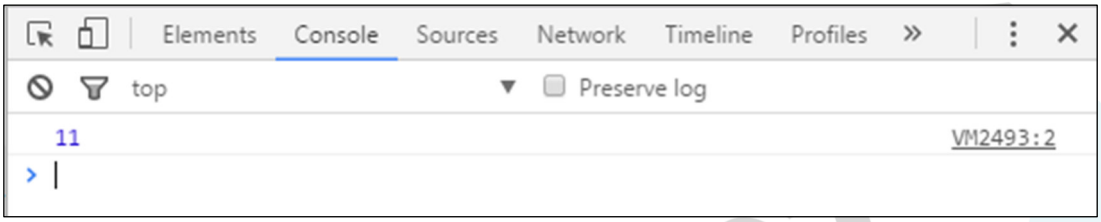
2. **windows.alert()**: Muestra un **mensaje de alerta** para mostrar información o datos. Es una ventana modal que se ejecuta en primer plano, detiene el flujo del programa y no deja manipular nada en segundo plano. Se puede escribir sin '***window'*** ya que depende del objeto principal dentro de una web que siempre es ***'window'***. En las páginas web actuales se intenta no abusar en exceso de este recurso debido al bloqueo que realiza sobre la web.

```
<script>

      alert(5 + 10);

</script>
```



3. **console.log**(): Se utiliza con propósitos de depuración. Crea un objeto del tipo Console() que nos permite imprimir un mensaje en la consola en cualquier momento sin interrumpir la ejecución del código. Es necesario abrir el navegador en modo de consola (tecla F12). Esta herramienta es muy útil para mostrar mensajes cuando se produce un error en la aplicación sin necesidad de mostrar esa información a través de la web.

```
// Muestra el mensaje a través de la consola el navegador

<script>

      console.log(5 + 6);

</script>
```



4. **console.info()**: Es similar a console.log() y se utiliza para mostrar mensajes de información.

5. **console.warm()**: Muestra información de advertencia presentándola en color amarillo.

6. **console.error():** Muestra información de error presentándola en color rojo. Para

7. **prompt**: Captura a través de una ventana modal la información que desea introducir el usuario. El valor capturado se almacena en una variable y se convertirá en otro tipo si fuese preciso ya que el valor recibido será **siempre de tipo texto** (*string*).

   Si intentamos asignar un valor a una variable que no está creada se mostrará el valor de '*Undefined*' (no definida) como resultado mostrado por pantalla. Si el prompt se cancela el valor devuelto por la ventana será 'null'.

```
<variable que almacena el dato> =
prompt (<msg a mostrar en pantalla>, <valor inicial por defecto>)

// 'prompt' captura órdenes por teclado y las guarda en variables.

Ejemplo: nombre = prompt('Cual es tu edad:','20');
```

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

      var nombre = prompt('Introduce tu nombre:','Pep');

      var ciudad = prompt('Introduce tu ciudad:','BCN');
```

```
        document.write('Hola ' + nombre);
        document.write(' Vivo en ' + ciudad);
    </script>
```

8. **innerHTML**: Escribe en un documento HTML o inserta contenido dentro de una etiqueta HTML. Esta propiedad permite incrustar etiquetas HTML siempre que se encuentren en formato texto ya que las reconoce y las interpreta como tal.

```
    // Escribe el número 10 dentro del elemento HTML especificado
    <script>
        document.getElementById("resul").innerHTML = 10;
    </script>

    // Escribe el nombre 'Homer' incluyendo codigo HTML en el mensaje.
    <script>
        document.getElementById("resul").innerHTML = "<h1>Homer</h1>";
    </script>
```

VARIABLES Y OPERADORES

Los lenguajes de programación como JavaScript (pseudo-lenguajes de alto nivel) ejecutan el código de programación de forma secuencial, es decir, una línea tras otra. La sintaxis de JavaScript no difiere demasiado con otros lenguajes, pero si la forma de escribirlo y representarlo en pantalla.

Las variables son elementos que permiten almacenar información para poder ser utilizada a lo largo de la ejecución del programa. Las variables en **lenguajes dinámicos** como Javascript se adaptan al tipo de valor introducido, es decir, si introducimos un número la variable es de tipo Number, y si introducimos un texto la variable será de tipo String. Además una misma variable puede cambiar de tipo sin verse afectada.

Las variables en los denominados **lenguajes estáticos** (C, Java u otros) se debe especificar que tipo de valor almacenará la variable y no podrá introducirse un valor de otro tipo ya que provocaría un error en tiempo de ejecución.

Los diferentes tipos de variables que se utilizan en Javascript son:

Tipo de dato	Descripción	Ejemplo básico
number	Valor numérico (enteros, decimales, etc...)	42
string	Valor de texto (cadenas de texto, carácteres, etc...)	'MZ'
boolean	Valor booleano (valores verdadero o falso)	true
undefined	Valor sin definir (variable sin inicializar)	undefined
function	Función (función guardada en una variable)	function() {}
object	Objeto (estructura más compleja)	{}

Los lenguajes como Javascript donde no hay que ser tan específico con la declaración de las variables se denominan lenguajes **débilmente tipados**, en cambio, otros lenguajes que son mucho mas estrictos con las declaraciones de los diferentes elementos y variables se consideran **fuertemente tipados**.

Las variables se pueden escribir de diferentes formas en función de la forma seleccionada por el usuario. No hay una forma mejor que otra pero siempre es recomendable seguir alguno de los sistemas estandarizados entre desarrolladores.

Nombre	Descripción	Ejemplo
camelCase	Primera palabra, minúsculas. El resto en minúsculas, salvo la primera letra. La más utilizada en Javascript.	**precioProducto**
PascalCase	Ídem a la anterior, pero todas las palabras empiezan con la primera letra mayúscula. Se utiliza en las Clases.	**PrecioProducto**
snake_case	Las palabras se separan con un guion bajo y se escriben siempre en minúsculas.	**precio_producto**
kebab-case	Las palabras se separan con un guion normal y se escriben siempre en minúsculas.	**precio-producto**

Variables

Para poder realizar rutinas o procedimientos es necesario utilizar **variables** que son los elementos que nos permiten **almacenar valores** y realizar cálculos. Algunas características de las variables son:

- Las variables son elementos que permiten **almacenar información de diferente tipo**. Disponen de un espacio de memoria (RAM) reservada en el ordenador, y almacenan valores que podrán cambiar durante la ejecución del programa.
    - Valores enteros (5, 8, 25, etc…). Numero solo parte entera.
    - Valores reales (4.21, 8.35, etc...). Numero con parte entera y decimal.
    - Cadenas de caracteres ('Juan', 'Pedro', 'Ana'). Las dobles comillas son aceptadas.
    - Valores Lógicos (true o false).

- **Las variables se declaran** escribiendo la palabra reservada **var, let, const** (siempre en minúsculas) delante del nombre de la variable. En su inicialización, una variable es de un tipo u otro cuando se le asigna un valor. Es recomendable que en la declaración de las variables estas sean **inicializadas** (darles un valor) para evitar problemas o errores en tiempo de ejecución.

```
var sexo = "mujer";    /* Texto siempre entre dobles comillas */
var talla = 1.72;      /* Para poner la coma es un punto */
```

- La declaración de variables en Javascript es siempre con las palabras reservadas, '**var**', **'let'**, o **'const'**. Otros lenguajes tienen declaraciones específicas si los valores almacenados serán números (int, float, double, etc..) o textos (char, string, etc..).

- Los nombres de las variables deben ser **representativos de su contenido** (nombre, edad, etc…) para realizar un mejor seguimiento del código en modo depuración.

- Los valores de las variables pueden ser **fijos [literales] o variables [variables]**. Hay variables que NO modifican su valor y otras variables que lo van modificando con el desarrollo del programa. Las variables fijas llamadas constantes se suelen escribir en mayúsculas para diferenciarlas del resto de las variables del código.

- Las variables comienzan por una letra o por guion bajo (_) o un signo dólar $, **nunca por un número** ya que la podría confundir por un valor numérico.

- Las palabras propiamente reservadas por el lenguaje **no se pueden utilizar como nombres de variables**. Por ejemplo: *var, break, if, while, case, for, function*, foreach, etc… no serían validas.

- La variable **debe estar inicializada con un valor** en el momento de su declaración. Si no se declara esta adopta el valor '**undefined**'. Este valor puede provocar problemas en tiempo de ejecución por no estar definida cuando se necesite para realizar operaciones.

```
var edad; // Si no se inicializa adopta el valor 'undefined'
var edad = 35; // Inicializada correctamente
```

- Los guiones (-) NO se aceptan para declarar nombres de variables. Nombres como '*dia-alta*' o '*fecha-nac*' no serían válidos.

- Los nombres de variables diferencian entre mayúsculas y minúsculas [**case sensitive**]. En palabras compuestas se recomienda utilizar el sistema [**camel case**] o [**lower camel case**] donde la primera letra de la palabra compuesta es en mayúsculas, como por ejemplo *'DiaSemana'*, *'NombrePersona'*, *'colorCamisa'*, *'tipoAnimal'*.

Ejemplo 1: Se crea un script con una declaración de variables que están inicializadas para posteriormente mostrarlas por la pantalla del navegador. En este caso el nombre de la variable dentro del método write no se escribe con comillas simples, porque sino provocaría que se mostrase el nombre de la variable, pero no su valor.

```
<html>
  <head> … </head>
  <body>
    <script type="text/javascript">
        var nombre = 'Juan';
        var edad = 10;
        var altura = 1.92;
        var casado = false;

        document.write(nombre);     //siempre en minúsculas
        document.write('<br>');
        document.write(edad);
        document.write('<br>');
        document.write(altura);
        document.write('<br>');
        document.write(casado);
    </script>
  </body>
  </html>
```

Declaración e inicialización de las variables del procedimiento. Siempre se declaran al principio.

Muestra en pantalla los valores de las variables e introduce una etiqueta HTML de salto de línea.

Ejemplos de declaración de variables

```
var nombreMarca;      // Declaración de la variable

var nombreMarca = "Audi";  // Asigna un valor a variable declarada

var persona = "Pepe", price = 200; // Asignación multivariable 1 línea

document.write(persona); // Mostrar por la pantalla del navegador
```

```
var precio1 = 5;

var precio2 = 6;

var total = precio1 + precio2;

document.write(total);
```

Las variables '**precio1**' y '**precio2**' se inicializan y almacenan un valor. La variable 'total' guarda el cálculo para finalmente mostrarlo por pantalla del navegador.

Las variables de escape se utilizan para poder utilizar caracteres reservados como si no lo fuesen o para realizar determinadas acciones específicas sobre el documento JavaScript.

Si se quiere incluir...	Se debe incluir...
Una nueva línea	\n
Devolver cursor inicio línea	\r
Un tabulador	\t
Una comilla simple	\'
Una comilla doble	\"
Una barra inclinada	\\

El siguiente ejemplo contiene la variable *'dato'* que almacena un texto. Al ser mostrado a través de la ventana del navegador, la línea se convierte en dos debido a las variables de escape que contiene dentro de la declaración fuerzan el ajuste de línea.

```
var dato = "El siguiente texto \r\n se escribirá en dos líneas";
```

Ámbito de las variables

Las variables, igual que en todos los lenguajes de programación, tienen un ámbito de acción (**scope**). El ámbito de las variables se determina en el momento de su declaración. Por tanto, es posible disponer de los siguientes tipos:

Variables locales

Las variables declaradas dentro de una función se consideran locales y solo se puede acceder a ellas desde dentro de la misma función. Es posible utilizar variables con el mismo nombre en diferentes funciones. Se eliminan de memoria cuando se dejan de utilizar. Si no se utiliza el prefijo ***'var', 'let' o 'const'*** para la declaración de la variable esta se convertirá en una variable global.

```
function miFuncion() {
    var coche = "Volvo"; // Solo afecta dentro de la función
    }
```

Variables globales

Las variables globales están declaradas fuera de las funciones de JavaScript. Esto significa que todas las funciones y procedimientos pueden acceder a esta variable y manipularla. En la POO no se recomienda en absoluto trabajar utilizando variables globales.

```
<script>
    var coche = "Volvo"; // Variable global declarada externamente
    myFunction();       // Llamada a la función

    function myFunction() {
        document.getElementById("demo").innerHTML =
        "El coche es" + coche; // Utiliza la variable global
    }
</script>
```

Las variables que no se hayan declarado correctamente se convierten de forma automática en variables *'globales'* incluso aun estando dentro de una función. En HTML todas las variables globales pertenecen al objeto ventana (**window**).

```
function miFuncion() {
    coche = "Volvo"; // Variable mal declarada, ahora es global
}
window.coche = "Mercedes";
```

Si dentro de una función declaramos una variable sin introducir el prefijo '***var***' entonces se convierte en una variable global de forma automática que se puede acceder desde cualquier parte de la aplicación.

Las variables en Javascript pueden ser de tres tipos en función de su ámbito de acción (scope).

var: Es de ámbito más genérico adquiriendo la característica de variable global que ***puede ser accesible desde cualquier parte del código***. Este tipo de variable se pasa a scopes descendientes o herederos y pueden ser modificadas por las funciones de la aplicación. Una variable de este tipo declarada en una función tiene un scope que afecta exclusivamente a toda la función.

let: Es un tipo de declaración que ***afecta exclusivamente al bloque de código donde ha sido declarada***, por ejemplo, un *for*, *while*, *if*, etc… Una vez se sale del bloque de código la variable desaparece liberando espacio en memoria. Las variables de este tipo al ser declaradas no se ven afectadas por el hoisting (alzamiento en la declaración) como le sucede con las de tipo 'var'.

const: Este tipo de declaración ***NO permite la reasignación de valores***, es decir, es una variable de solo lectura. Se acostumbran a escribir en mayúsculas para diferenciarlas de los otros tipos de variables. Existe la posibilidad de cambiar el valor de la variable si este formase parte de un objeto, pero es un caso que sale de la finalidad de esta variable. Las variables de tipo 'const' no se ven afectadas por el hoisting.

Tipos de datos

Los valores en JS se almacenan siempre en variables mediante una de las palabras reservadas '**var**', '**let**' o '**const**'. Cada variable puede almacenar un tipo diferente de valor y los más habituales son:

```
var edad = 16;        // Número entero
var descuento = 15.25;     // Número con decimales
var nombre = "Pep";    // String o cadena de texto
var x = true;         // Booleano verdadero, no se pone entre comillas
var x = false;        // Booleano falso. True y false ya son valores
var cars = ["Saab", "Volvo", "BMW"];  // Array o matriz de valores

// Objeto que almacena 4 propiedades.
var person = { Nombre:"Pep", Ape1:"Lopez", Edad:50, Sexo:"H" };

var person;     // Devuelve 'undefined', sin valor y sin tipo
var person="";  // Valor vacío, no tiene nada que ver con 'undefined'
var person = null;   // Valor nulo, pero tipo sigue siendo un objeto
```

Función **typeof()**: Es un método global de JavaScript que **permite averiguar el tipo de dato** que almacena una variable. En muchos casos **typeof()** resulta insuficiente porque trabajamos con objetos o elementos más avanzados. En estos casos la sentencia **constructor.name** nos proporciona el tipo de constructor que utiliza.

```
typeof "John"          // Devuelve tipo "string"
typeof 3.14            // Devuelve tipo "number"
typeof true            // Devuelve tipo "boolean"
typeof [1,2,3,4]                 // Devuelve tipo "object"
typeof {nombre:'John', edad:34} // Devuelve tipo "object"
typeof function myFunc(){}      // Devuelve tipo "function"

console.log(nomPersona.constructor.name)  // Devuelve tipo "string"
```

Función **instanceOf()**: Indica si un valor es una instancia a un tipo de objeto.

Comentarios

En JavaScript es posible **comentar líneas de código o bloques de código**. Para comentar líneas se realiza con la doble contra barra mientras que para comentar un bloque se realiza con asterisco-contra barra.

Los **comentarios son imprescindibles** para explicar el código que se está realizando. Esto es necesario para poder leer y entender el código que se ha realizado en otro momento.

```
var x = 5;       // Comentario para una línea (Comentario de línea)

/* Párrafo comentado que no se ejecutará (Comentario de bloque)
document.getElementById("myH").innerHTML = "My First Page";*/
```

Operadores

Son los símbolos o caracteres que permiten realizar operaciones matemáticas. Existen diferentes tipos de operadores en función del uso.

- Operadores aritméticos

  Para realizar operaciones matemáticas básicas.

Operador	Descripción
+	Suma (o concatena)
-	Resta
*	Multiplicación
**	Potencia (ES6)
/	División
%	Resto división (módulo)
++	Incrementar en 1
--	Decrementar en 1

Los **operadores tienen prioridad** de unos respecto a otros. La multiplicación y la división tienen preferencia respecto a la suma y la resta. En caso de igualdad de prioridad entre dos operadores se ejecutan de izquierda a derecha.

```
var x = 5, var y = 2;   // Declaración múltiple misma línea.
var z = x + y;          // Suma, return 7
var m = x * y;          // Multiplicación, return 10
var x ++;               // Incremento de 1, return 6
```

- Operadores de asignación

Para dar valor a las diferentes variables.

Operador	Ejemplo (método abreviado)	Lo mismo que... (método expandido)
=	x = y	x = y (asignación)
+=	x += y	x = x + y
-=	x -= y	x = x – y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

Ejemplos

```
var x = 10;
x += 5;
```

Se asigna el valor 10 a la variable 'x' y se le suma 5. Resultado final 15.

```
x = 5 + 5;      // Asigna la suma como valor numérico (55)
y = "5" + 5;    // Si hay texto el cálculo devuelve texto ('55')
z = "Hello" + 5; // Texto y numero devuelve texto ('hello5')

var txt1 = "Programación";
var txt2 = "JavaScript";
var txt3 = txt1 + " " + txt2; // Concatena dos textos con espacio

txt1 += "en JavaScript"; //Concatena texto con operador asignación

var x = 10;
x *= 5;    // El resultado final será 50 (10*5)

var x = 10;
x /= 5;    // Devuelve la parte entera de la división (10/5)=2
```

- Operadores de comparación

Para realizar comparaciones entre variables.

Operador	Descripción
==	Igual a
===	Igual valor y igual tipo
!=	Diferente (no igual)
!==	Diferente valor o diferente tipo
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que
?	Condicional abreviado (op. ternario)

Ejemplos

```
var x = 10;       // Declaración de la variable.

x == "10";        // Devuelve True, mismo valor sin valorar tipo.

x > 20;           // Devuelve False

x != 20;          // Devuelve True

x >= 10;          // Devuelve True

x < 10;           // Devuelve False

x !== "10";       // Devuelve False, mismo valor diferente tipo.

x === 20;         // Devuelve False, diferente valor mismo tipo.
```

- Operadores de lógicos

Operador	Descripción
&&	Condicional Y. Todas las condiciones se deben cumplir
||	Condicional O. Una o más condiciones se deben cumplir
!	Negación. Para negar el resultado de una condición.

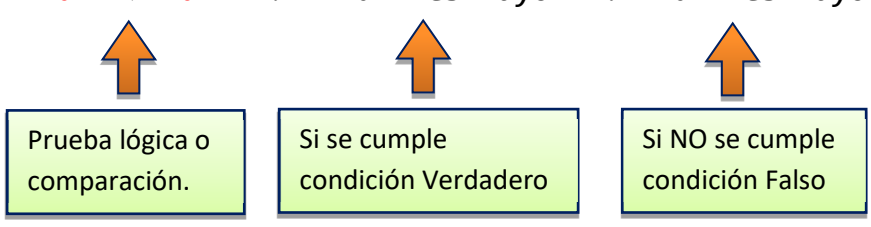
```
// Si las 2 condiciones se cumplen entra en el bucle.

   if (num1 > num2  &&  num1 > num3) {

      document.write('el mayor es el '+ num1);

   }


// Si una de las 2 condiciones o las 2 se cumplen entra en el bucle.

   if (num1 > num2  ||  num1 > num3)  {

       document.write('El mayor es el '+ num1);

   }
```

- Operador ternario

El operador ternario (o condicional) permite realizar una operación condicional de forma más abreviada y aprovechar el resultado para asignárselo a una variable.

La sintaxis del operador ternario siempre devuelve algo que se almacena en una variable:

```
   < Condición ? True : False >

// En este caso la variable 'resultado' almacena el resultado de la
condición utilizando el operador ternario.

   var resultado = num1 > num2  ?  "Num1 es mayor" : "Num2 es mayor";
```



USE STRICT

Para no tener problemas con variables locales que se declaran sin querer como globales (entre otros posibles errores) existe la directiva '*use strict*' que obliga, entre otras cosas, a declarar todas las variables dentro de las funciones ya que sino produce un error desde (ES6).

El modo estricto se puede aplicar al código Javascript por completo o a una función. El 'modo estricto' **es una variante del lenguaje que es menos permisiva con ciertos comportamientos** de tal forma que posibles warnings en modo normal se convierten en errores de Javascript en 'modo estricto'. Los programadores profesionales utilizan el *'strict mode'* para minimizar los posibles errores al programar.

La directiva de modo estricto emplea las palabras claves '***use strict***' entre comillas simples o dobles seguido de punto y coma. Los navegadores modernos implementan los análisis de modo estricto. La razón de disponer la directiva entre comillas y con un punto y coma es para que navegadores antiguos pasen por alto el string.

Ejemplo: El siguiente código tiene declarado antes de empezar a ejecutar ningún código el método '***use strict***'. Este puede estar en un ámbito global o dentro de una función.
De esta forma la variable 'x' que se encuentra declarada dentro de la función **imprimir()** dará como resultado un error en tiempo de ejecución porque ahora es obligatorio declararla.
Sin esta línea de código en modo estricto la variable '*x*' simplemente hubiese sido considerada como una variable global y por tanto podría ser accesible desde cualquier zona del código.

```
<script type="text/javascript">
 "use strict";
/* Modo estricto que obliga a declarar todas las */
      function imprimir(){
           // Da error, mal declarada, está en modo 'use strict'
           x = 100;   document.write( x +'<br>');
      }
 imprimir();  document.write( x +'<br>');
</script>
```

ESTRUCTURAS DE CONTROL DE FLUJO

Una estructura secuencial es un algoritmo que dispone de **un único hilo o camino para ejecutarse**. Un ejemplo de estructura secuencial sería pedir datos al usuario y mostrarlos por pantalla. Este algoritmo tiene un único recorrido, por tanto, se considera una estructura secuencial.

Los operadores aritméticos (+, - , * , / , %) permiten realizar operaciones matemáticas. Los valores capturados por la aplicación se deberán convertir a número ya que por defecto la información es texto. Las funciones que permiten la conversión de texto a número son:

- **parseInt()**: Convierte un texto que representa un número entero a número entero.
- **parseFloat()**: Convierte un texto que representa un número decimal a número decimal.
- **Number()**: Convierte cualquier número (entero o decimal), a valor numérico. Es el método que se recomienda utilizar porque engloba los dos anteriores y además tiene otras funciones extra.

```
<script type="text/javascript">
  var num1, num2; //declaración de las variables
  num1 = prompt('Ingrese primer valor:','');
  num2 = prompt('Ingrese segundo valor:','');

  // Realiza la suma convirtiendo los valores a enteros
   var suma = parseInt(num1)+ parseInt(num2);
  // Realiza el producto convirtiendo valores a números
   var producto = Number(num1) * Number(num2);

  document.write('La suma es:' + suma);
  document.write('El producto es:' + producto);
</script>
```

Solicita valor con 'prompt' y lo guarda en la variable.

Escribe el texto y el contenido de las variables por pantalla.

IF (Estructura condicional Simple)

Las condiciones son las sentencias mediante las cuales se puede determinar que un proceso realice una acción u otra en función de un criterio establecido.

Este tipo de estructuras se utilizan cuando tenemos más de una solución o camino para ejecutarse nuestra aplicación JavaScript. Un ejemplo condicional puede ser si un alumno ha sacado más de un 5 entonces está aprobado, sino está suspendido.

Para utilizar estructuras condicionales es necesario conocer los operadores de comparación:

- **>** Mayor.
- **>=** Mayor o igual.
- **<** Menor.
- **<=** Menor o igual.
- **!=** Distinto o Diferente.
- **!==** Diferente valor y tipo de dato.
- **==** Igual (de comparación, porque un solo igual es asignación).
- **===** Igualdad (mismo valor y mismo tipo).

En JavaScript, como en otros lenguajes, aparece la instrucción **if**. Se escribe entre paréntesis y ejecuta el código si se cumple la condición. Además, la función **if** se escribe siempre en minúsculas y tiene llaves de apertura y de cierre. En caso contrario JavaScript devolverá un error en tiempo de ejecución.

El siguiente ejemplo muestra una estructura condicional con IF.

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

  var nombre = prompt('Ingrese nombre:','');
  if (nota >= 5){    // Si la nota es superior o igual a 5.
    document.write(nombre +' está aprobado con un '+ nota);
  }
</script>
```

Las estructuras condicionales compuestas son iguales que las simples solo que se les añade la opción '***else'*** (sino). De esta forma si no cumple una condición automáticamente realiza el cálculo que haya dentro de '***else'***. En este tipo de condición se ejecuta una de las dos opciones '***if'*** o '***else'*** pero en ningún caso se ejecutarán las dos condiciones a la vez. '**else**' también dispone de sus llaves de apertura y cierre.

```
if (<condición>) { <Instruccion(es)> }
else { <Instruccion(es)> }
```

La estructura ***else if*** también es válida en este tipo de estructuras y puede repetirse tantas veces como sea necesaria.

```
function calculoNota(){
    var nota;
    var resultado;

    if (nota < 5) {
        resultado = "Suspendido";
    } else if (nota < 8) {
        resultado = "Notable";
    } else if (nota < 10) {
        resultado = "Excelente";
    }else {
        resultado = "Excelente nota";
    }

    // Escribe el valor en una caja de la web mediante innerHTML.
    document.getElementById("parrafo").innerHTML = resultado;}
```

Declaración de las variables.

Condicionales para determinar el valor de la variable resultado.

Las estructuras **condicionales anidadas** se caracterizan por incluir instrucciones dentro de funciones. Siguiendo el caso anterior podemos utilizar una instrucción ***if*** dentro de otra instrucción ***if*** quedando de la siguiente forma. Utilizando este sistema podemos dar salida a tantos resultados como nos convenga.

```
if (<condición>)
{
  <Instruccion(es)>
}
else
{
        if (<condición>) { <Instrucción(es)> }
        else { <Instrucción(es)>}
}
```

BUCLE WHILE // DO WHILE

El tipo de estructura ***while*** es un bucle de código fuente que se ejecuta siempre mientras se esté cumpliendo la condición que se haya especificado. Si en la primera pasada la condición no se cumple automáticamente salta al siguiente código sin tener en cuenta el bucle ***while***.

```
while ( x <= 100){
    document.write(x);
    document.write('<br>');
    x=x+1;
  }
```

El bucle se repite 'mientras' la variable x sea inferior o igual a 100. El resultado se muestra en el navegador. En este tipo de estructura puede darse el caso que no entre ninguna vez.

```
//El bucle carga un texto durante 10 veces empezando desde cero.
while (i < 10) {
    texto += "El número es " + i;
    i++; }

// Carga el valor de texto en la caja con id 'parrafo' de HTML
document.getElementById("parrafo").innerHTML = texto;
```

El tipo de estructura ***do ... while*** es un bucle de código que funciona exactamente igual que ***while*** pero la diferencia radica en que en esta segunda forma **como mínimo pasa una vez por el bucle** hasta que llega a la condición para ser evaluado el criterio.

```
do {
    valor = prompt('Ingrese valor:',''); // Pide valor por teclado
    valor = Number(valor); // Convierte a entero el num introducido
    suma = suma + valor; // Suma el valor al total de la suma.
    x = x+1;    // Aumenta la variable 'contadora' x en una unidad.
} while (x <= 10)
```

BUCLE FOR

Este tipo de bucle realiza un número de pasadas determinadas por el código. En este caso sabemos el número de veces que se pasará por dicho código. El bucle ***for*** es utilizado frecuentemente para recorrer arrays y dispone de tres "parámetros".

1. El valor inicial del bucle. Por regla general suele ser cero este valor. Se puede omitir este valor y adoptará el valor que almacene en memoria. Es recomendable introducirlo dentro del bucle ***for*** y iniciarlo con el valor deseado para evitar sustos inesperados.

2. El valor condicional del bucle. Es un criterio o condición que determinará .

3. El incremento o decremento del bucle. Es una variable del tipo contador que al llegar a un valor determinado hace no vuelva a entrar en el bucle.

La inclusión y ejecución de la sentencia ***break;*** en cualquier parte del bucle provoca la salida inmediatamente del bucle. Esta sentencia puede ser útil para "hacer saltar" del bucle durante la ejecución del código cuando se ha encontrado un registro en concreto y de esta forma no será necesario tener que recorrer el resto del elemento.

Ejemplo 1:

```
// El bucle concatena un texto durante 5 veces empezando desde cero.
// Valor inicial:0, Mientras que 'i' sea < 5, incrementando de 1 en 1.
for (i = 0; i < 5; i++) { texto += "El número es " + i + "<br>";}

// Carga el valor de texto en la caja con id 'parrafo' de HTML
document.getElementById("parrafo").innerHTML = texto;
```

El número es 0
El número es 1
El número es 2
El número es 3
El número es 4

Resultado: Escribe en la caja el resultado que se ha ido cargando en cada pasada del bucle **for.**

Ejemplo 2:

```
// Declaración de la variable con sus valores. Esto es un array.
var coches = ["BMW", "Volvo", "Saab", "Ford", "Audi"];

// Con la propiedad 'length' sabemos que el tamaño del array es 5.
// El bucle prepara el texto para mostrarlo en la caja de HTML
for (i = 0; i < coches.length; i++) {
    text += coches[i] + "<br>";}

// Carga el valor de texto en la caja con id 'párrafo' de HTML
document.getElementById("parrafo").innerHTML = texto;
```

ESTRUCTURAS SWITCH

Este tipo de estructuras se utilizan cuando se disponen de muchas posibles soluciones y como alternativa a la función **if**. Se utiliza cuando sabemos el valor exacto que podemos recibir, es decir, la comparación da como resultado verdadero o falso.

- **switch:** inicia este tipo de estructura. Se le debe especificar un valor mediante el cual realizará la comparación con cada uno de los casos.
- **case:** se utiliza para determinar el código a ejecutar si existe coincidencia para cada uno de los diferentes posibles resultados.
- **default:** determina el código que se ejecutará si ninguno de los casos posibles se ha podido ejecutar por falta de coincidencia.
- **break:** permite salir de la sentencia switch en el caso que alguno de los códigos haya sido ejecutado. Con esta palabra se libera al navegador de tener que comparar con todos los casos.

```
switch (valor) {
    case 1: document.write("uno");
        break; // Hace saltar del switch
    case 2: document.write("dos");
        break;
    case 3: document.write("tres");
        break;
    case 4: // Se pueden poner 2 casos sobre el mismo código
    case 5: document.write("cuatro o cinco");
        break;
    default:document.write("Introduce un numero entre 1 y 4.");
}
```

En el siguiente ejemplo los diferentes casos son cadenas de texto.

```
switch (color) {
    case "blanco": document.write("color blanco");
          break;
    case "negro": document.write("color negro");
          break;
    case "rojo": document.write("color rojo");
          break;
    default:document.write("Color no es blanco ni negro ni rojo");
}
```

Instrucciones de transferencia de control

En algunas ocasiones **los bucles se deben interrumpir**. JavaScript ofrece utiliza dos palabras reservadas para detener la ejecución de bucles y condicionales. Aparentemente realizan la misma acción (interrumpir) pero lo ejecutan con alguna diferencia.

- **continue:** Esta instrucción interrumpe el ciclo actual y avanza hasta el siguiente ciclo dentro del mismo bucle sin ejecutar el resto del código del ciclo. En este caso el bucle principal se sigue ejecutando, pero concretamente en ese ciclo las líneas de código que hay a continuación de *'continue'* se pasan por alto hasta el siguiente ciclo.

- **break:** Esta instrucción interrumpe el bucle. Todas las instrucciones restantes y los ciclos pendientes se ignoran saliendo del bucle para continuar con el resto del código.

```
for (i = 0; i < coches.length; i++){
  var marca = coches[i];
   if (marca == "Seat") {
        continue;
     }
   text += coches[i] + "<br>";
}
```

El bucle FOR recorre un array. Si en alguna posición del array el valor es "Seat", salta de ciclo y pasa a la siguiente posición del array sin ejecutar el resto del código. Si fuese una instrucción **break** en vez de **continue**, directamente saltaría del bucle FOR.

Este tipo de saltos dentro de código en POO (programación orientada objetos) no son recomendables ni del todo bien vistos ya que rompen la lógica de la programación. Es decir, el código se tiene que ejecutar de forma secuencial de principio a fin sin utilizar atajos o puertas traseras para evitar ejecutar determinadas zonas de código.

Una utilidad práctica que puede dar sentido a este tipo de saltos es en aplicaciones donde los bucles disponen de muchas líneas de código y de esta forma se consigue ganar tiempo de respuesta y velocidad en la ejecución de la aplicación.

> Ejemplo: Un algoritmo para consultar datos personales de clientes que contenga un bucle de 500 líneas de código y con más de 100.000 pasadas.