



# JAVASCRIPT

ISABEL SUAREZ RUZ

## TABLA DE CONTENIDO

<b>1</b>	<b>CONCEPTO DE ALGORITMO .....</b>	<b>3</b>
<b>2</b>	<b>CONCEPTO DE PROGRAMA .....</b>	<b>4</b>
2.1	FASES PARA LA ELABORACIÓN DE UN PROGRAMA .....	5
<b>3</b>	<b>ESTRUCTURA DE UN FICHERO HTML .....</b>	<b>6</b>
3.1	¿QUÉ SE NECESITA PARA CREAR UNA PÁGINA WEB? .....	6
<b>4</b>	<b>¿QUÉ ES JAVA SCRIPT?.....</b>	<b>7</b>
4.1	SINTAXIS BÁSICA EN JAVASCRIPT .....	7
<b>5</b>	<b>ELEMENTOS DE UN LENGUAJE DE PROGRAMACIÓN .....</b>	<b>9</b>
5.1	LOS DATOS.....	9
5.1.1	DECLARACION DE UNA VARIABLE.....	10
5.1.2	TIPOS DE VARIABLES.....	11
5.1.2.1	NUMÉRICAS .....	11
5.1.2.2	CADENAS DE TEXTO .....	12
5.1.2.3	BOOLEANOS.....	12
5.1.2.4	VARIABLES NULAS .....	12
5.2	EXPRESIONES: TIPOS Y OPERADORES.....	13
5.2.1	OPERADOR DE ASIGNACIÓN .....	13
5.2.2	NUMÉRICAS: OPERADORES ARITMÉTICOS.....	13
5.2.2.1	OPERADORES MATEMÁTICOS .....	14
5.2.2.2	INCREMENTO Y DECREMENTO .....	14
5.2.3	EXPRESIONES LÓGICAS: OPERADORES RELACIONALES Y LÓGICOS .....	16
5.2.3.1	OPERADORES LÓGICOS.....	16
5.2.3.1.1	Negación .....	16
5.2.3.1.2	AND .....	16
5.2.3.1.3	OR.....	17
5.2.3.2	OPERADORES RELACIONALES.....	17
5.2.4	JERARQUÍA DE LOS OPERADORES .....	18
<b>6</b>	<b>INSTRUCCIONES Y TIPOS .....</b>	<b>19</b>
6.1	INSTRUCCIONES SECUENCIALES .....	19
6.1.1	<b>INSTRUCCIONES DE ASIGNACIÓN.....</b>	<b>19</b>
6.1.1.1	CONTADORES .....	20
6.1.1.2	ACUMULADORES.....	20
6.1.1.3	INTERRUPTORES .....	20
6.1.2	<b>INSTRUCCIONES DE LECTURA Y ESCRITURA:ENTRADA Y SALIDA DE LA INFORMACIÓN ....</b>	<b>20</b>
6.2	INSTRUCCIONES DE CONTROL DE FLUJO.....	24
6.2.1	<b>ESTRUCTURAS CONDICIONALES .....</b>	<b>24</b>
6.2.1.1	ESTRUCTURA CONDICIONAL SIMPLE .....	24
6.2.1.2	ESTRUCTURA CONDICIONAL DOBLE .....	26
6.2.1.3	ESTRUCTURA CONDICIONAL MÚLTIPLE .....	28
6.2.2	<b>ESTRUCTURAS REPETITIVAS .....</b>	<b>32</b>
6.2.2.1	ESTRUCTURA WHILE.....	33
6.2.2.2	ESTRUCTURA DO...WHILE .....	36
6.2.2.3	FOR .....	37
<b>7</b>	<b>FUNCIONES.....</b>	<b>39</b>
7.1	DECLARACIÓN DE UNA FUNCIÓN.....	40
7.2	ÁMBITO DE LA FUNCIÓN: VARIABLES LOCALES Y GLOBALES .....	41

7.3	ENTRADA DE VALORES: ARGUMENTOS O PARÁMETROS .....	42
7.4	LLAMADA O INVOCACIÓN A LA FUNCIÓN .....	42
7.5	RETURN: RETORNO DE UN VALOR: VALOR DE SALIDA. ....	43
7.6	UTILIZACIÓN DE FUNCIONES PARA VALIDAR VARIABLES .....	44
7.7	ASPECTOS A TENER EN CUENTA.....	45
<b>8</b>	<b>ESTRUCTURA DE DATOS ARRAYS.....</b>	<b>46</b>
8.1	DECLARACIÓN DEL ARRAY.....	46
8.2	ACCESO A LOS ELEMENTOS DEL ARRAY .....	47
8.3	PROPIEDADES Y MÉTODOS. HERRAMIENTAS PARA MANIPULAR EL ARRAY .....	48
8.3.1	<b>PROPIEDAD LENGTH</b> .....	48
8.3.2	<b>MÉTODO PUSH</b> .....	48
8.3.3	<b>MÉTODO UNSHIFT</b> .....	49
8.3.4	<b>MÉTODO pop</b> .....	49
8.3.5	<b>MÉTODO shift</b> .....	50
8.3.6	<b>MÉTODO CONCAT</b> .....	50
8.3.7	<b>MÉTODO SPLICE</b> .....	51
8.3.8	<b>MÉTODO SLICE</b> .....	52
8.3.9	<b>MÉTODO JOIN</b> .....	53
8.3.10	<b>MÉTODO SORT Y REVERSE</b> .....	53
<b>9</b>	<b>PROGRAMACIÓN ORIENTADA A OBJETOS (POO) .....</b>	<b>54</b>
9.1	¿QUÉ SON LAS CLASES? .....	56
9.2	¿QUE ES UN OBJETO? .....	56
9.3	ACCEDER A LAS PROPIEDADES Y MÉTODOS .....	56
9.4	CLASES PREDEFINIDAS EN JAVASCRIPT .....	58
9.4.1	<b>clase string</b> .....	58
9.4.2	<b>CLASE DATE</b> .....	61

Distintas fuentes de Internet. Entre ellas:

PÁGINA LIBROSWEB

PÁGINA DESARROLLO WEB

CANAL PILDORAS INFORMÁTICAS

DIGITAL LEARNING

<https://www.digitallearning.es/intro-programacion-js/objetos-propiedades-metodos.html>

UNIWEBSIDAD

<https://uniwebsidad.com/libros/javascript/capitulo-1>

## 1 CONCEPTO DE ALGORITMO

Se define un **Algoritmo** como un conjunto de acciones o secuencia de operaciones (instrucciones), escritas en lenguaje coloquial, que ejecutadas en un determinado orden resuelven el problema. No existe una única solución, sino que hay diferentes algoritmos para un mismo problema. Escogeremos la opción que sea más eficiente.

**Ejemplo:** considera el algoritmo que se elaboraría para el problema o situación de levantarse todas las mañanas para ir al trabajo:

- Acción 1.** Salir de la cama
- Acción 2.** quitarse el pijama
- Acción 3.** ducharse
- Acción 4.** vestirse
- Acción 5.** desayunar
- Acción 6.** arrancar el automóvil para ir al trabajo o tomar transporte.

Pero esta secuencia de acciones puede refinarse más, y para cada acción definir más subacciones, condicionar o repetir la acción u acciones hasta que se cumpla una determinada condición:

- Acción 1.** Si toca el despertador
  - a. Salir de la cama
  - b. Quitar el pijama
  - c. Ducharse
  - d. Vestirse
  - e. Desayunar
  - f. Arrancar el coche
- Acción 2.** Si no, seguir durmiendo.

Esto es lo que ocurriría un día normal. Pero para el caso de todos los días de la semana sería:

- Acción 1.** Repetir desde lunes hasta viernes
  - a. Si toca el despertador
    - i. Salir de la cama
    - ii. Quitar el pijama
    - iii. Ducharse
    - iv. Vestirse
    - v. Desayunar
    - vi. Arrancar el coche
  - b. Si no, seguir durmiendo.

## 2 CONCEPTO DE PROGRAMA

Pero para que el algoritmo anterior se convierta en un programa, además de codificar las instrucciones en un lenguaje de programación, habría que proporcionarle unos datos de entrada para que nos devuelva el resultado, es decir, los datos de salida.

En este sentido, se define un **Programa** como un conjunto de Instrucciones (Algoritmo) y Datos para resolver el problema.

**Ejempl:** pensemos en el ejemplo de la suma de números. Para empezar, los **datos de entrada** que primeramente tendremos que proporcionarle al ordenador son esos dos números, y el ordenador nos proporcionará como **dato de salida** el resultado de la suma.

Para entendernos con un ordenador se utilizan los lenguajes informáticos. Existen dos grandes grupos de lenguajes a la hora de escribir un programa: **Los compilados** y **Los interpretados**

Cuando usamos lenguajes **compilados** seguimos los siguientes pasos:

- 1º). Escribir el programa (código fuente),
- 2º). Compilar el programa, es decir, traducir al lenguaje interno del ordenador.
- 3º). Se obtiene así un nuevo fichero que es el que se ejecuta tantas veces como se desee, que se guarda con la extensión .exe y .com

En resumen, un programa compilado se traduce una vez y se ejecuta cuantas veces se desee. A este grupo pertenecen los lenguajes tradicionales como C, Pascal, JAVA....

El otro grupo es el de los lenguajes **interpretados**, en donde:

- 1º). Se escribe (código fuente).
- 2º). El ordenador va leyendo cada instrucción, la traduce y la ejecuta.

Es decir, es necesario traducir el programa cada vez que se ejecuta. A este grupo pertenece Basic, el Perl y los llamados scripts como JAVASCRIPT y Vbscript. Los programas escritos en estos dos últimos lenguajes son interpretados por los navegadores de Internet como Internet Explorer, Netscape Navigator, Opera, ...

Cabe destacar una diferenciación entre JAVA Y JAVASCRIPT porque son completamente diferentes. Solo comparten parte del nombre. En definitiva:

**JAVA:** **lenguaje de programación de propósito general.** Para crear aplicaciones de escritorio, para ejecutarse en una página web (applets) en un navegador, aplicaciones que se ejecutan en un servidor es decir, en un ordenador remoto, aplicaciones para ejecutarse en dispositivos móviles. Es un lenguaje que ha de compilarse, es decir, que crea un fichero adicional ejecutable.

**JAVASCRIPT: lenguaje de propósito concreto.** Lenguaje que tiene un objetivo concreto, en este caso que se ejecutan en un navegador. Está orientado a la web y se va a ejecutar siempre en un navegador. Se ejecuta de forma local, en tu ordenador no en un servidor, esto significa que es de respuesta inmediata. Es un lenguaje interpretado, se ejecuta línea a línea. Se encarga de crear interactividad a la página web. Dota de efectos visuales dinámicos a la página (efectos a la hora de abrir una imagen, efectos de transición de una página web a otra, efectos a la hora de abrir un menú,...)

## 2.1 FASES PARA LA ELABORACIÓN DE UN PROGRAMA

A la hora de construir un programa se realiza en una serie de fases, y son:

- 1º). Análisis previo o evaluación del problema: Estudiar el problema en general y ver que parte nos interesa. Mediante entrevistas a los clientes se analiza lo que se quiere programar.
- 2º). Análisis detallado: Ver que datos hay que proporcionarle al programa y cuáles serán los resultados en pantalla, así como las posibles condiciones o restricciones.
  - **Entrada de datos**: son los datos iniciales que va a necesitar el programa para resolver el problema. Normalmente se ejecutan a través de dispositivos de entrada datos en el ordenador como teclado, tarjetas digitalizadoras... Son instrucciones de lectura.
  - **Acciones de un algoritmo O INSTRUCCIONES**: Parte en la que se resuelve el problema usando los datos de entrada.
  - **Salida de datos**: Mostrar en un dispositivo de salida los resultados de las acciones anteriormente realizadas. Son acciones de escritura.
- 3º). Diseño del algoritmo: Diseñar la solución.
- 4º). El programa: Codificación del algoritmo en un lenguaje de programación.
- 5º). Pruebas: Ver si el programa hace lo que queríamos. El modo más normal de comprobarlo es realizando la TRAZA de un programa, es decir, usar datos significativos que abarquen todo el rango de valores.

**Ejemplo:** Diseña el algoritmo para sumar dos números.

### ANALISIS DEL PROBLEMA:

- **Datos Entrada:** a,b
- **Instrucciones:** sumar a y b
- **Datos de Salida:** Mostrar el resultado de la suma

### DISEÑO DEL ALGORITMO

1. Introducir dos números a y b por medio de un dispositivo de entrada de datos en el Ordenador.
2. Calcular la suma de a mas b
3. Mostrar en un dispositivo de salida el resultado de la suma.

En nuestro caso, es decir con JavaScript, nuestra misión será decirle al ordenador que cuando presente nuestra página web al visitante haga cosas como poner en la página la fecha del día, hacer que una imagen se mueva de un lado a otro, responder de una determinada forma a la pulsación del ratón, etc. Pero antes de poder realizar esto debemos aprender las nociones básicas de programación de cualquier lenguaje.

### 3 ESTRUCTURA DE UN FICHERO HTML

El lenguaje HTML está basado en el uso de etiquetas (tags), por eso se dice que es un lenguaje de etiquetas, que tienen una estructura de la forma:

`<etiqueta>Hola</etiqueta>`

Las siglas HTML significan HyperText Markup Language, es decir, Lenguaje de Marcas de Hipertexto. Estas etiquetas deben abrirse `<etiqueta>` y cerrarse `</etiqueta>`. De esta forma, la estructura de un documento (o página) HTML, básicamente consta de cuatro pares de etiquetas:

```
<HTML>
  <HEAD>
    <TITLE>Título de la página</TITLE>
    ...
  </HEAD>

  <BODY>
    Aquí iría el contenido de la página: texto, enlaces,
    fotos,...
  </BODY>
</HTML>
```

El contenido de un fichero HTML podemos subdividirlo en 2 partes fundamentales: la **cabecera** (HEAD) y el **cuerpo** (BODY). El código JavaScript lo insertaremos en el cuerpo mayoritariamente, aunque en la cabecera también se podrán insertar ciertas funciones, que serán llamadas desde el cuerpo.

- ⇒ El par `<HTML>` y `</HTML>` Determina que un fichero sea HTML
- ⇒ El par `<HEAD>` y `</HEAD>` Determina la cabecera del fichero HTML, que puede contener un título, y donde también se incluirá el código JavaScript
- ⇒ El par `<TITLE>` y `</TITLE>` Encierra el "título": frase de texto que aparecerá en el marco del navegador (primera línea), al ejecutar el fichero HTML
- ⇒ El par `<BODY>` y `</BODY>` Encierra el contenido de la página html, es decir lo que se visualizará en el navegador.

#### 3.1 ¿QUÉ SE NECESITA PARA CREAR UNA PÁGINA WEB?

Para crear una PAGINA WEB necesitamos hacer uso de tres tecnologías:

- 1º). **HTML**: para crear la **estructura** de una página web
- 2º). **CSS**: para darle aspecto o **apariciencia** de un página web
- 3º). **JAVASCRIPT**: para agregar interactividad a la página web, es decir, dotar de cierta **inteligencia** a la página web, como por ejemplo, que nos reconozca cuando entramos en la página mediante un usuario y una contraseña y sepa quiénes somos.

## 4 ¿QUÉ ES JAVA SCRIPT?

JavaScript es un lenguaje de programación creado por la empresa Netscape (creadora de uno de los navegadores más conocido)

Es el lenguaje de programación más utilizado en Internet para añadir interactividad a las páginas Web, ya que crear un documento HTML es crear algo de carácter estático. La página se carga simplemente, pero no podemos INTERACTUAR con ella. Así pues, como solución a este problema, nace JavaScript. De esta forma, podremos añadir un menú desplegable, mover una imagen por la Pantalla, validar el texto entrado por el usuario, confirmaciones...

Con JavaScript no pueden construirse programas independientes, sólo pueden escribirse scripts(guiones) que funcionarán en el entorno de una página Web, interpretado por un explorador, de ahí la importancia de conocer para que explorador escribimos los guiones.

Un programa en JavaScript se integra en una página Web (entre el código HTML) y es el navegador el que lo interpreta. Como se explicó anteriormente, JavaScript es un lenguaje interpretado, no compilado (no se genera ningún tipo de fichero objeto o .exe). Es por ello, que las sentencias de JavaScript se integran dentro del código HTML.

Para programar en JavaScript sólo necesitamos un editor de texto (podemos utilizar el Bloc de Notas del Windows o un editor de Java), guardarlo con extensión .HTM y tener un navegador (utilizaremos Microsoft Internet Explorer) para ejecutarlo.

¿Por qué aprenderemos JavaScript y no otro lenguaje de programación?:

- ⇒ Es moderno.
- ⇒ Es sencillo.
- ⇒ Es barato: sólo necesitamos un editor de textos (el "Bloc de Notas" está incluido en Windows) y un navegador (es gratuito, ya sea "Internet Explorer" o "Mozilla").
- ⇒ Es visual: permite la moderna "programación visual" (ventanas, botones, colores, formularios, etc.).

### 4.1 SINTAXIS BÁSICA EN JAVASCRIPT

Como se ha indicado anteriormente, un programa "JavaScript" se escribe integrado en una página HTML, por lo tanto no es más que un fichero de texto que contiene una serie de pares de **tags** (o pares de etiquetas) correspondientes a la página Web (como mínimo el par: <HTML>, </HTML>), además del par de tags característico de un programa JavaScript. El fichero resultante se ha de grabar necesariamente con la extensión HTM (característica de una página HTML).

Un programa "JavaScript" no es más que una secuencia de órdenes, encerradas entre las etiquetas, que deben terminar en punto y coma. Para hacer que un documento HTML incluya instrucciones en JavaScript se debe utilizar el par de etiquetas <SCRIPT> de la siguiente forma:

```
<SCRIPT LANGUAGE="JavaScript">
```

*Código Javascript*

```
</SCRIPT>
```



El atributo LANGUAGE hace referencia a la versión de JavaScript que se va a utilizar en dicho script.

Otro atributo de la directiva script es SRC, que puede usarse para incluir un archivo externo que contiene JavaScript y que quiere incluirse en el código HTML.

```
<script language="JavaScript" src ="archivo.js">
</script>
```

## NORMAS DE ESCRITURA EN JAVASCRIPT

- ⇒ Los comentarios deben empezar con el símbolo // si son de una sola línea, o empezar con los símbolos /\* y terminar con \*/ si son de más de una línea.
- ⇒ Cada línea de código termina en ;
- ⇒ Se distingue entre mayúsculas y minúsculas
- ⇒ Las llaves { } permiten agrupar código

**Ejemplo:** El siguiente programa escribe en pantalla la cadena de texto "Hola Mundo".

```
<HTML>
  <SCRIPT LANGUAGE='JavaScript'>
    alert('¡Hola Mundo!');
  </SCRIPT>
</HTML>
```



Al hacer clic en el [Aceptar] de la ventana "alert", se acaba el programa JavaScript (se encuentra el tag </SCRIPT>) y continua ejecutándose la página HTML.

## 5 ELEMENTOS DE UN LENGUAJE DE PROGRAMACIÓN

Como hemos dicho anteriormente para poder realizar un programa no solo necesitamos las **instrucciones** propias del lenguaje de programación, sino que además necesitamos los **datos**, que se le proporcionarán al programa y que a su vez se mostraran en la pantalla. Estos datos son lo que se denominan **variables** y **constantes**.

Para poder operar con las **variables**, a su vez necesitaremos los distintos **operadores**, que formaran unas **expresiones** matemáticas.

Así pues, a continuación se muestran los diferentes elementos que formaran el lenguaje de programación:

<b>Variables</b>	Etiquetas que se refieren a un valor cambiante.
<b>Operadores</b>	Pueden usarse para calcular o comparar valores. Ejemplo: pueden sumarse dos valores, pueden compararse dos valores...
<b>Expresiones</b>	Cualquier combinación de variables, operadores, y declaraciones que evalúan a algún resultado. Ejemplo: <code>intTotal=100; intTotal &gt; 100</code>
<b>Sentencias</b>	Una sentencia puede incluir cualquier elemento de la gramática de JavaScript. Las sentencias de JavaScript pueden tomar la forma de condicional, bucle, o manipulaciones del objeto. La forma correcta para separarlas es por punto y coma, esto sólo es obligatorio si las declaraciones múltiples residen en la misma línea.

### 5.1 LOS DATOS

Un Dato es un objeto o elemento que tratamos a lo largo de diversas operaciones. Se distinguen las **variables** y las **constantes**, y se caracterizan por:

- ⇒ Un nombre que los diferencia del resto.
- ⇒ Un tipo que nos determina las operaciones que podemos hacer con ese dato.
- ⇒ Un valor que puede variar o no a lo largo de la operación.

**Ejemplo:** *queremos sumar dos números cualesquiera que sean. Para ello, en vez de usar dentro del programa los dos números en cuestión, les damos un nombre: a y b. De esta forma podremos utilizar el programa para sumar cuántas veces queramos.*

Las variables y constantes no son más que posiciones de memoria a las que se les da un nombre y en las que se almacena valor asociado a dicha variable o constante y cuando el programa las use, irá a esa zona de memoria a buscar su valor. La diferencia principal entre las dos es:

- ⇒ **Constantes:** Tienen un valor fijo que se le da cuando se define la constante y que ya no puede ser modificado durante la ejecución. Las constantes pueden llevar asociados un nombre o no,

si no lo llevan, se llaman literales. Su valor hay que darlo al definir la constante y ya no puede cambiar a lo largo de la ejecución.

Ejemplo:  $\pi=3,1416$

⇒ **Variables:** El valor puede cambiar durante la ejecución del algoritmo, pero nunca varía su nombre y su tipo. Antes de usar una variable hay que definirla o declararla, al hacerlo hay que dar su nombre y su tipo. El nombre que le damos tiene que ser un nombre significativo. Tiene que empezar por una letra, y el tamaño depende del lenguaje. El valor de la variable si al declararla no se la inicializa, en algunos lenguajes toma una por defecto. En cualquier caso el valor de la variable podemos darle uno inicial, introducirlo por teclado o podemos ir variándolo a lo largo de la ejecución.

Gracias a las variables es posible crear "*programas genéricos*", es decir, programas que funcionan siempre igual independientemente de los valores concretos utilizados.

Si no existieran variables, un programa que suma dos números podría escribirse como:

Suma = 3 + 6

Como podemos observar es poco útil puesto que sólo sirve para sumar el 3 y el 6. Sin embargo, utilizando variables podremos reutilizar el programa para la suma de distintos números:

suma = a+b

Los elementos a y b son **variables** que almacenan los valores que utiliza el programa. El resultado se calcula siempre en función del valor que le daremos a lo largo del programa a dichas variables.

### 5.1.1 DECLARACION DE UNA VARIABLE

Declarar variables consiste en definir y de paso informar al sistema de que vamos a utilizar una variable.

Las variables en JavaScript se crean mediante la palabra reservada **var**. La palabra reservada **var** solamente se debe indicar al definir por primera vez la variable, lo que se denomina **declarar una variable**.

Una variable se puede declarar en JavaScript, de dos formas:

**Forma Explícita:** var nombre Variable;  
**Forma Implícita:** var nombre Variable= valor;

En el último caso se dice que la variable ha sido **inicializada**.

De esta forma, el ejemplo anterior se puede realizar en JavaScript de la siguiente manera:

```
var a = 2;
var b = 1;
var suma = a+b;
```

En JavaScript no es obligatorio inicializar las variables, ya que se pueden declarar por una parte y asignarles un valor posteriormente. Por tanto, el ejemplo anterior se puede rehacer de la siguiente manera:

```
var a;
var b;
```

```
a = 3;
b = 2;
var suma = a + b;
```

También se pueden utilizar variables que no se han definido anteriormente mediante la palabra reservada `var`. El ejemplo anterior también es correcto en JavaScript de la siguiente forma:

```
var a = 3;
var b = 2;
suma = a + b;
```

No obstante, se recomienda declarar todas las variables que se vayan a utilizar.

### 5.1.2 TIPOS DE VARIABLES

En una variable podemos introducir varios tipos de información, por ejemplo texto, números enteros o reales, etc. A estas distintas clases de información se les conoce como **tipos de datos**. A diferencia de otros lenguajes, en Javascript no es necesario declarar las variables especificando el tipo de dato que contendrán, será el propio interprete el que le asignará el tipo apropiado. (Esto es así para seguir la filosofía de diseño de Javascript que indica que se realizan programas pequeños y que la idea es lograr que el programador realice los scripts de la manera más rápida posible). Es al asignarle un valor cuando pasa a ser de uno u otro tipo, según el dato que albergue.

Existen 6 tipos de datos:

<b>Números</b>	Enteros o coma flotante.
<b>Lógicos o Booleanos</b>	True o False.
<b>Cadenas de caracteres</b>	Abarca al conjunto finito y ordenado de caracteres que reconoce la computadora (letras, dígitos, caracteres especiales, ASCII). Los tipos de datos cadena deben ir delimitados por comillas simples o dobles.
<b>Objetos</b>	Obj = new Object();
<b>Nulos</b>	Null
<b>Indefinidos</b>	Un valor indefinido es el que corresponde a una variable que ha sido creada pero no le ha sido asignado un valor.

Una variable puede cambiar de tipo de datos a lo largo de la ejecución del código fuente. Es decir, que puede contener al principio por ejemplo un valor numérico y más adelante tomar valor de tipo texto. Esto es porque Javascript realiza conversión de datos.

#### 5.1.2.1 NUMÉRICAS

Se utilizan para almacenar valores numéricos enteros o decimales (llamados *float* en inglés). En este caso, el valor se asigna indicando directamente el número entero o decimal. Los números decimales utilizan el carácter `.` (punto) en vez de `,` (coma) para separar la parte entera y la parte decimal:

```
var pi = 3,1416;
var total = 35,16; // variable tipo decimal
```

---

#### 5.1.2.2 CADENAS DE TEXTO

Se utilizan para almacenar caracteres, palabras y/o frases de texto. Para asignar el valor a la variable, se encierra el valor entre comillas dobles o simples, para delimitar su comienzo y su final:

```
var nombre = "Isabel";  
var Opcion = 'c';
```

Si por ejemplo el propio texto contiene comillas simples o dobles, se debe encerrar el texto con las comillas (simples o dobles) que no utilice el texto:

```
var texto1 = "Una frase con 'comillas simples' dentro";  
var texto2 = 'Una frase con "comillas dobles" dentro';
```

El ejemplo anterior que contenía comillas simples y dobles dentro del texto se puede rehacer de la siguiente forma:

```
var texto1 = 'Una frase con \'comillas simples\' dentro';  
var texto2 = "Una frase con \"comillas dobles\" dentro";
```

---

#### 5.1.2.3 BOOLEANOS

Las variables de tipo *boolean* o *booleano* también se conocen con el nombre de variables de tipo lógico. Aunque una variable de tipo *boolean* solamente puede tomar dos valores: `true` (verdadero) o `false` (falso).

```
var alumna_matriculada = false;
```

---

#### 5.1.2.4 VARIABLES NULAS

Cuando una variable no contiene ningún valor, su contenido es nulo

## 5.2 EXPRESIONES: TIPOS Y OPERADORES

Los **operadores** permiten manipular el valor de las variables, realizar operaciones matemáticas con sus valores y comparar diferentes variables.

Una **expresión** es una combinación de constantes, variables, operadores, paréntesis y nombres especiales (nombres de funciones estándar), de cuya evaluación resulta un único valor.

**Ejemplo:**  $(a+b)-c$  siendo a, b, y c variables

Toda expresión tiene asociada un tipo que se corresponde con el tipo del valor que devuelve la expresión cuando se evalúa, por lo que habrá tantos tipos de expresiones como tipos de datos. Habrá expresiones numéricas, alfanuméricas y lógicas.

**Ejemplo:** en el ejemplo anterior si a y b son números enteros, pero c es un número decimal el resultado de la operación será un número decimal

### 5.2.1 OPERADOR DE ASIGNACIÓN

El operador de asignación se utiliza para guardar un valor específico en una variable. El símbolo utilizado es `=` (no confundir con el operador `==` que es el operador de igualdad, que se verá más adelante):

```
var a = 3;
suma = a + 2
```

A la izquierda del operador, siempre debe indicarse el nombre de una variable. A la derecha del operador, se pueden indicar variables, valores, condiciones lógicas, etc.

El proceso de asignación se realiza en 2 fases:

- 1º). Se evalúa la expresión de la parte derecha de la asignación obteniéndose un único valor.
- 2º). Se asigna ese valor a la variable de la parte izquierda.

**Ejemplo:** `SUMA = A + B` (Primero se realiza la suma del contenido de las variables A y B y por último, se almacena en la variable suma.)

### 5.2.2 NUMÉRICAS: OPERADORES ARITMÉTICOS

Son los que se utilizan en las expresiones numéricas (una combinación de variables y/o constantes numéricas con operadores aritméticos y que al evaluarla devuelve un valor numérico).

Operador	Significado
+	Suma
-	Resta
*	Multiplicación
/	División

%	Resto división
++	Incremento
--	Decremento

### 5.2.2.1 OPERADORES MATEMÁTICOS

Los operadores definidos son: suma (+), resta (-), multiplicación (\*) y división (/). Ejemplo:

```
var a = 10;
var b = 5;

resultado = a / b; // resultado = 2
resultado = 3 + a; // resultado = 13
resultado = b - 4; // resultado = 1
resultado = a * b; // resultado = 50
```

Además tenemos otro operador "módulo", que se indica mediante el símbolo %, que calcula el resto de la división entera de dos números. Si se divide por ejemplo 10 y 5, la división es exacta y da un resultado de 2. El resto de esa división es 0, por lo que módulo de 10 y 5 es igual a 0.

Sin embargo, si se divide 9 y 5, la división no es exacta, el resultado es 1 y el resto 4, por lo que módulo de 9 y 5 es igual a 4.

Los operadores matemáticos también se pueden combinar con el operador de asignación para abreviar su notación:

```
var a = 5;
a += 3; // a = a + 3 = 8
a -= 1; // a = a - 1 = 4
a *= 2; // a = a * 2 = 10
a /= 5; // a = a / 5 = 1
a %= 4; // a = a % 4 = 1
```

### 5.2.2.2 INCREMENTO Y DECREMENTO

Se utilizan para incrementar o decrementar en una unidad el valor de una variable numérica.

El operador de incremento se indica mediante el prefijo ++ en el nombre de la variable. El resultado es que el valor de esa variable se incrementa en una unidad.

```
var a = 5;
++a; // a = 6
```

El anterior ejemplo es equivalente a:

```
var a = 5;
a = a + 1; // a = 6
```

De igual forma, el operador decremento (indicado como un prefijo -- en el nombre de la variable) se utiliza para decrementar el valor de la variable:

```
var a = 5;
--a; // a = 4
```

El anterior ejemplo es equivalente a:

```
var a = 5;  
a = a - 1; // a = 4
```

Los operadores de incremento y decremento no solamente se pueden indicar como **prefijo** del nombre de la variable, sino que también es posible utilizarlos como **sufijo**, pero entonces se comporta de distinta manera: Si el operador `++` se indica como prefijo del identificador de la variable, su valor se incrementa **antes** de realizar cualquier otra operación. Si el operador `++` se indica como sufijo del identificador de la variable, su valor se incrementa **después** de ejecutar la sentencia en la que aparece.

```
var a = 5;  
var b = 2;  
c = a++ + b; // c = 7, a = 6, es decir, primero se suma 5+2=7 y después se  
incrementa a en 1 siendo a=6.
```

```
var a = 5;  
var b = 2;  
c = ++a + b; // c = 8, a = 6, es decir primero se incrementa a en 1 siendo a=6,  
después se suma 6+2=8
```



### 5.2.3 EXPRESIONES LÓGICAS: OPERADORES RELACIONALES Y LÓGICOS

Una expresión lógica es aquella que sólo puede devolver dos valores (True o False). Los valores que pueden aparecer en una expresión lógica son de 2 tipos: lógicos y relacionales.

La particularidad de las expresiones lógicas es que mientras en una expresión numérica para devolver un valor numérico los operandos solo pueden ser números, en una expresión lógica los operandos no tienen porque ser booleanos aunque se devuelva un valor booleano. Esto es lo que ocurre cuando en la expresión lógica utilizamos operadores relacionales con lo cual se obtienen valores lógicos o booleanos a partir de otros que no lo son. En cambio cuando los operadores son lógicos los operandos obligatoriamente también tienen que ser lógicos.

**Ejemplo:**      100 < 99      → FALSO  
                   100 > 99      → VERDADERO  
                   Número comprendido dentro del rango 1 al 100: (n>1 y n<100)

Operadores relacionales (comparación)	
>	Mayor
<	Menor
==	Igual
!=	Distinto
>=	Mayor o igual
<=	Menor o igual
Operadores lógicos evaluar por más de una condición al mismo tiempo	
&&	AND ( Y lógico)
	OR ( O lógico)
!	NOT (NO lógico)

#### 5.2.3.1 OPERADORES LÓGICOS

Los operadores lógicos se utilizan para tomar decisiones sobre las instrucciones que debería ejecutar el programa en función de ciertas condiciones.

##### 5.2.3.1.1 NEGACIÓN

Se utiliza para obtener el valor contrario al valor de la variable. Se obtiene prefijando el símbolo !

```
var visible = true;
alert(!visible); // Muestra "false" y no "true"
```

##### 5.2.3.1.2 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 como se indica la tabla:

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

```
var a = true;
var b = false;
resultado = a && b; // resultado = false
```

```
a = true;
b = true;
resultado = a && b; // resultado = true
```

#### 5.2.3.1.3 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`:

variable1	variable2	variable1    variable2
true	true	true
true	false	true
false	true	true
false	false	false

```
var a = true;
var b = false;
resultado = a || b; // resultado = true
```

```
valor1 = false;
valor2 = false;
resultado = a || b; // resultado = false
```

#### 5.2.3.2 OPERADORES RELACIONALES

Los operadores relacionales son: mayor que (`>`), menor que (`<`), mayor o igual (`>=`), menor o igual (`<=`), igual que (`==`) y distinto de (`!=`).

```
var a = 3;
var b = 5;
```

```
resultado = a > b; // resultado = false
resultado = a < b; // resultado = true
resultado = a == b; // resultado = true
resultado = a != b; // resultado = false
```

#### 5.2.4 JERARQUÍA DE LOS OPERADORES

Es muy importante saber el orden jerárquico de los operadores, por ejemplo  $2 + 3 * 5$  da como resultado 17 y no 25 ya que el operador del producto se aplica primero. Si se tienen dudas lo mejor es usar paréntesis para que el orden de los operandos quede lo más claro posible. En el caso anterior no es lo mismo  $(2 + 3) * 5$  que  $2 + (3 * 5)$ .

El orden de prioridad de los operadores de mayor a menor es el siguiente:

⇒ `!, ++, --`

⇒ `*, /, %`

⇒ `+, -`

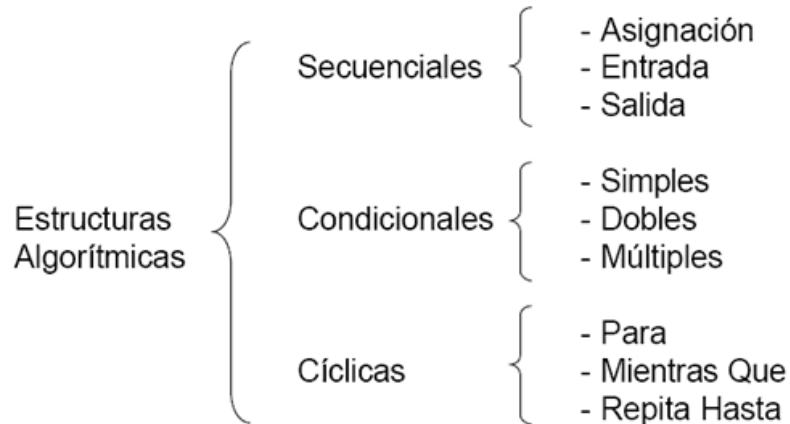
⇒ `<, <=, >, >=, ==, !=`

⇒ `&, ^, |`

⇒ `&&, ||`

## 6 INSTRUCCIONES Y TIPOS

Las instrucciones disponibles para escribir un programa dependen del lenguaje de programación utilizado. Existen instrucciones -o acciones- básicas que se pueden implementar de modo general en cualquier programa y que soportan todos los lenguajes de programación. Estas son:



### 6.1 INSTRUCCIONES SECUENCIALES

Es cuando una instrucción sigue a otra en secuencia, es decir, la salida de una instrucción es la entrada de la siguiente. Son instrucciones de asignación, de entrada y de salida como hemos visto en los ejemplos anteriores.

#### 6.1.1 INSTRUCCIONES DE ASIGNACIÓN

El operador de asignación se utiliza para guardar un valor específico en una variable. El símbolo utilizado es = (no confundir con el operador == que es el operador de igualdad que se verá más adelante):

```
var numero1 = 3;
suma = numero1 + 2
```

A la izquierda del operador, siempre debe indicarse el nombre de una variable. A la derecha del operador, se pueden indicar variables, valores, condiciones lógicas, etc.

El proceso de asignación se realiza en 2 fases:

- 3º). Se evalúa la expresión de la parte derecha de la asignación obteniéndose un único valor.
- 4º). Se asigna ese valor a la variable de la parte izquierda.

*Ejemplo:* SUMA = A + B (**Primero se realiza la suma del contenido de las variables A y B y por último, se almacena en la variable suma.**)

Cuando asignamos una expresión a una variable debemos distinguir entre los siguientes tipos de asignaciones:

- Contadores
- Acumuladores
- Interruptores

### 6.1.1.1 CONTADORES

Los contadores se utilizan en las *instrucciones repetitivas* o *bucles* con la finalidad de contar las veces que se ejecuta el bucle atendiendo a una finalidad.

Primero, se deberá inicializar el valor de la variable contador, y después mediante otra instrucción de asignación la incrementaremos o decrementaremos. Recordemos que una variable es una posición de memoria.

**Ejemplo:**

```
Cont=0
...
Cont=cont + 1 # incrementamos en 1 el valor de la variable utilizando el valor anterior
```

### 6.1.1.2 ACUMULADORES

Son variables cuyo valor se incrementa o decrementa en una cantidad determinada. Es decir, acumula el resultado de la anterior operación. También debe inicializarse previamente.

**Ejemplo:**

```
Suma=0
...
Suma=suma + n
```

### 6.1.1.3 INTERRUPTORES

Son variables que pueden tomar el valor verdadero o falso y sirve para controlar la entrada en un bucle y estructuras selectivas. También tienen que inicializarse a verdadero o falso.

**Ejemplo:**

```
valor=verdadero
...
Si no verdadero entonces
```

## 6.1.2 INSTRUCCIONES DE LECTURA Y ESCRITURA: ENTRADA Y SALIDA DE LA INFORMACIÓN

En la operación de lectura una variable recibe un valor, pero este valor no resulta de evaluar ninguna expresión, sino que el valor lo vamos a leer de un dispositivo de entrada (teclado, tableta digitalizadora,...). Se trata de una ventana que pide al usuario que introduzca datos al programa. Esta ventana, además, podrá ser aceptada o cancelada. Si es cancelada devolverá el valor Null.

**Nombre\_Variable=prompt (Texto\_mensaje, valor\_por\_defecto)**

**Ejemplo:** num=prompt("Introduce un número entero:"). Escribe en pantalla el texto Introduce un número y espera que se le introduzca desde teclado un número. A continuación guarda ese número en la variable Num para ser utilizado más adelante.

De esta forma, la función prompt nos permite introducir 'valores', dichos valores se han de guardar en variables, que previamente hemos de declarar con la palabra reservada 'var'

La operación de escritura lo que hace es mostrar el valor de una variable o mostrar el entrecomillado que se encuentra como argumento de la función. Esta operación la podemos realizar de dos formas:

- Mediante una ventana de alerta emergente, utilizando la función **alert**:

**alert (Texto)**

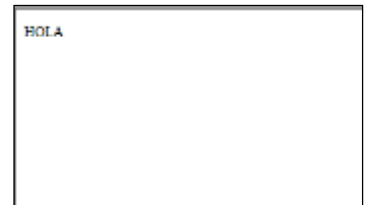
El mensaje que se muestra en pantalla solo da la opción de ACEPTAR.

**Ejemplo:** alert ("Hola") Muestra una ventana emergente con la palabra "HOLA".

b). Utilizando el comando de escritura en la página:

**document.write(texto)**

**Ejemplo:** document.write("Hola"). Escribe HOLA en la página

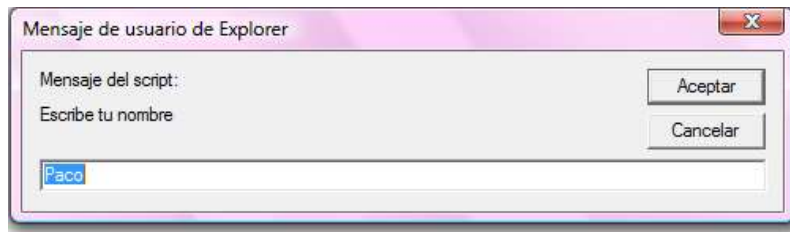


Si utilizamos el comando document.write("<br>") baja de línea en el navegador.

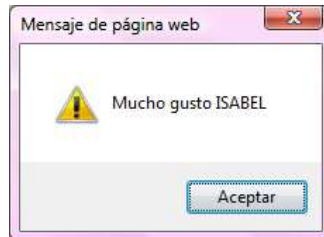
**EJEMPLO2:** Escribe el siguiente programa. Grábalo con el nombre Ejemplo2.htm

```
<HTML>
<SCRIPT LANGUAGE='JavaScript'>
// EJEMPLO2.HTM
    var nom;
    nom=prompt('Escribe tu nombre ','Paco');
    alert('Mucho gusto '+ nom);
</SCRIPT>
</HTML>
```

- ⇒ Primera y última línea: <HTML> y </HTML> página html mínima que necesitamos para incluir un programa JavaScript.
- ⇒ Segunda y penúltima líneas: <SCRIPT LANGUAGE=.....> y </SCRIPT>, es decir programa en JavaScript
- ⇒ Primera sentencia del programa: // EJEMPLO2.htm es un comentario indicando el nombre del programa. Única línea del programa que no es necesario acabarla con punto y coma.
- ⇒ var nom; Declaramos una variable de nombre nom
- ⇒ nom = prompt("Escribe tu nombre","Paco"); Aparece un recuadro con un mensaje y un campo donde podemos escribir algo; el mensaje corresponde a lo que escribimos en el primer argumento de la función prompt, encerrado entre comillas. El segundo argumento del prompt contiene el valor que aparece por defecto en el campo del cuadro de diálogo. El valor del prompt es nom, es decir lo que nosotros escribamos en el cuadro será el valor que tomará la variable nom. Si no escribimos nada y hacemos click en [Aceptar], el prompt, es decir la variable nom tomará el valor de Paco, porque es el valor que aparece por defecto.



- ⇒ `alert('Mucho gusto '+nom);` Aparece un cuadro con el mensaje 'Mucho gusto' y a continuación el valor de la variable 'nom', que será lo que hemos escrito en el primer cuadro que nos ha aparecido.



**Ejemplo:** Programa que calcule la suma de dos números introducimos por teclado y muestre el resultado de la suma en pantalla

## 1º). ANALISIS DEL PROBLEMA

- Entrada: a, b dos variables que representaran los dos números
- Instrucciones: sumar dos números
- Salida: Muestra en pantalla el resultado de la operación.

## 2º). DISEÑO

```
<HTML>
<SCRIPT LANGUAGE='JavaScript'>
// SUMA.HTM
    var a,b;
    a=prompt("Escribe un número","");
    b=prompt("Escribe otro numero","");
    alert("La suma es = "+(a + b));
</SCRIPT>
</HTML>
```

- ⇒ El programa no funciona porque la suma la interpreta como una concatenación (unión) del valor de a y de b dando como resultado 23 en vez de 5. A partir de este momento hemos de tener claro si los "prompts" corresponden a números enteros, decimales o de texto:

- ✓ Si "x" ha de ser un número entero escribiremos:  
`x = parseInt(prompt("Escribe un número entero",""));`
- ✓ Si "x" ha de ser un número decimal escribiremos:  
`x = parseFloat(prompt("Escribe un número entero o decimal",""));`
- ✓ Si "x" ha de ser una cadena de texto escribiremos:  
`x = prompt("Escribe el texto correspondiente","");`

Así, el programa quedaría de la siguiente forma:

```
<HTML>
<SCRIPT LANGUAGE='JavaScript'>
// SUMA.HTM
    var a,b;
    a=parseInt(prompt('Escribe un número',''));
    b=parseInt(prompt('Escribe otro numero',''));
    alert('La suma es = '+ (a + b));
</SCRIPT>
</HTML>
```

**Ejemplo:** Leer las longitudes de un rectángulo y calcular el área.

Para calcular el área de un rectángulo, se necesitan las medidas de la base y la altura, estas medidas serán leídas en dos variables. Las salidas serán el valor del área.

### 1º). ANALISIS DEL PROBLEMA

- Entradas: base, altura
- Instrucciones: cálculo del área
- Salidas: el valor del área mostrada en pantalla

### 2º). DISEÑO

```
<HTML>
<SCRIPT LANGUAGE='JavaScript'>
// EJEMPLO3.HTM
/*Programa que sirve para calcular
el área de un rectángulo , siendo la base y altura números decimales*/
    var base,altura;
    base=parseFloat(prompt('Escribe la base del Rectángulo',''));
    altura=parseFloat(prompt('Escribe la altura del Rectángulo',''));
    alert('El área del Rectángulo es = '+ (base*altura));
</SCRIPT>
</HTML>
```

- ⇒ Declaramos dos variables (var), que necesitamos para introducir la base y la altura del rectángulo, a través de dos 'prompts'
- ⇒ Todo lo que aparece escrito entre /\* y \*/ no es más que un comentario para el programador, igual que pasaba con las líneas que empezaban por //
- ⇒ La diferencia entre // y /\* \*/ está en que esta última forma de incluir un comentario, nos permite colocarlo de forma que ocupe más de una línea.
- ⇒ 'alert nos muestra el resultado del programa, que es simplemente el producto base \* altura

Si has ejecutado el programa una vez, para volverlo a ejecutar, no es necesario que cierres el navegador, solo pulsa F5.

### 3º). PRUEBA

base	altura	área	SALIDA EN PANTALLA
2	3	6	El área es 6



## 6.2 INSTRUCCIONES DE CONTROL DE FLUJO

**¿Qué significa control de flujo?** Ya sabemos que cuando el navegador empieza a leer el script para ejecutarlo lo hace en orden secuencial, o sea, una instrucción detrás de otra en un determinado orden. Pero a veces es necesario saltarse instrucciones, como por ejemplo, cuando en una página sólo pueden entrar determinadas personas. Para ello, se debe escribir una función que pida un dato (nombre, DNI, clave de acceso,...) de aquella persona que desea entrar en la página. Si es una persona autorizada muestra la página y si no lo es no la muestra. Cuando esto ocurre el programa no ha seguido un flujo lineal, unas veces ejecutará la parte de mostrar la página y otras no.

Otra caso es cuando queremos repetir una secuencia de acciones, como por ejemplo, que tu programa recorra todas las imágenes de tu página y vaya cambiando su contenido. Para ello, no se escribe el mismo código tantas veces queremos que se repita esa secuencia, sino que utilizamos una instrucción que le indica al programa que repita la secuencia hasta que se cumpla una condición para finalizarla.

Para todo ello, utilizamos las llamadas sentencias **de control del flujo de programa**. Son sentencias que permiten que se ejecuten condicionalmente algunos pasos (condicionales) o repetir una serie de instrucciones una y otra vez (bucles).

Estas instrucciones son del tipo:

- ⇒ "si se cumple esta condición, hazlo; si no se cumple, haz esto otro".
- ⇒ bien, "repite esto mientras se cumpla esta condición".

---

### 6.2.1 ESTRUCTURAS CONDICIONALES

Estas instrucciones se emplean para tomar decisiones en función de una condición. Se evalúa una condición, cuyo resultado será VERDADERO o FALSO. En función de ese resultado se ejecutarán un conjunto de instrucciones u otras. La condición se ejecuta una única vez.

Hay tres tipos de estructuras condicionales (simple, doble o múltiple):

---

#### 6.2.1.1 ESTRUCTURA CONDICIONAL SIMPLE

Evaluamos la condición, y si se cumple (es decir, si su valor es **verdadero**) se ejecutan todas las instrucciones que se encuentran dentro de las llaves {...}. Si la condición no se cumple (es decir, si su valor es **falso**), no se ejecuta ninguna instrucción contenida en {...} y el programa continúa ejecutando el resto de instrucciones que vayan a continuación.

```
if (condición)
{
    ...
}
```

*Ejemplo 2: Suma de dos números que sean positivos introducimos por teclado.*

Para saber si un número es positivo o negativo, debemos saber si es menor o mayor a cero. Si es mayor, el número es positivo y si es menor resulta negativo. Utilizamos *If...* para evaluar como es el número con respecto a cero y mostramos en pantalla los mensajes correspondientes en cada caso.

### 1º. ANÁLISIS

Introduce dos números:

4  
-5

### 2º. DISEÑO

```
<HTML>
<SCRIPT LANGUAGE='JavaScript'>
// SUMA POSITIVOS.HTM
    var a,b;
    a=parseInt(prompt('Escribe un número',''));
    b=parseInt(prompt('Escribe otro numero',''));
    if ((a>0) && (b>0))
    {
        alert('La suma es = '+ (a+b));
    }
</SCRIPT>
</HTML>
```

Solo le sumaran los números si los dos son a la vez positivos. En el momento que un número sea negativo no se realizará la suma.

### 3º. PRUEBA

Para probar una estructura condicional debemos poner todos los casos posibles que pueden tomar las variables:

a	b	suma	SALIDA EN PANTALLA
2	-3		Como b es negativo no realiza la suma y no muestrana en pantalla
-5	2		Como a es negativo no realiza la suma y no muestrana en pantalla
7	8	15	La suma es 15
-7	-2		Como son negativos no realiza la suma y no muestrana en pantalla

Para mejorar el algoritmo anterior deberíamos mostrar un mensaje de error en pantalla diciendo que no se pueden sumar porque al menos uno no es positivo. Para ello tenemos las instrucciones condicionales dobles.

### 6.2.1.2 ESTRUCTURA CONDICIONAL DOBLE

Se evalúa la **condición** y si es verdad se ejecutan el conjunto de acciones colocadas dentro del **"if"**, y si es falso se ejecutan el conjunto de acciones asociadas al **"else"**.

```
if(condición)
{
  ...
}
else
{
  ...
}
```

**Ejemplo1:** Introduce tu edad y comprueba si eres o no mayor de edad:

```
<SCRIPT LANGUAGE='JavaScript'>
  var edad;
  edad=prompt("Introduce tu edad:", "");
  if(edad >= 18)
  {
    alert("Eres mayor de edad");
  }
  else
  {
    alert("Eres menor de edad");
  }
</SCRIPT>
```

**Ejemplo2:** Sumar dos números positivos introducidos por teclado y mostrar en pantalla su resultado. Si alguno de ellos es negativo que muestre en pantalla un mensaje de error "ALGÚN NUMERO ES NEGATIVO"

```
<SCRIPT LANGUAGE='JavaScript'>
// SUMA POSITIVOS.HTM
var a,b;
a=parseInt(prompt('Escribe un número',''));
b=parseInt(prompt('Escribe otro numero',''));
if ((a>0) && (b>0))
{
  alert('La suma es = '+ (a+b));
}
else
{
  alert("Algun numero es negativo")
}
</SCRIPT>
```

También podemos incluir un **if** dentro de otro en el caso en que no sea suficiente evaluar los casos posibles. Esto se le llama una **estructura anidada**. Lo veremos con los siguientes ejemplos:

**Ejemplo 3** *Dados dos números, establecer cuál es mayor*

```

<SCRIPT LANGUAGE='JavaScript'>
    var a,b;
    a=parseInt(prompt('Escribe un número',''));
    b=parseInt(prompt('Escribe otro numero',''));
    if (a>b)
    {
        alert(a+'es mayor que '+b);
    }
    else
    {
        if (b>a)
        {
            alert(b+'es mayor que '+a);
        }
        else
        {
            alert("Los dos números son iguales");
        }
    }
}
</SCRIPT>

```

**Ejemplo 4.** *Clasifica por edades a las personas:*

```

<SCRIPT LANGUAGE='JavaScript'>
    var edad;
    edad=prompt("Introduce tu edad:", "");
    if(edad < 12)
    {
        alert("Eres un niño");
    }
    else
    {
        if(edad < 18)
        {
            alert("Eres un adolescente");
        }
        else
        {
            if(edad < 35)
            {
                alert("Aun sigues siendo joven");
            }
            else
            {
                alert("Piensa en cuidarte un poco más");
            }
        }
    }
}
</SCRIPT>

```

En este último ejercicio podemos observar que tenemos varios if...else anidados.

### 6.2.1.3 ESTRUCTURA CONDICIONAL MÚLTIPLE

Para que no resulte tedioso anidar varias estructuras `if..else`, podemos utilizar las “estructuras condicionales múltiples”. En estas estructuras, se evalúa los diferentes valores que puede tomar una **variable**. Según el **valor** (pueden ser números o caracteres) que la variable tenga en cada **caso** (`case`), se ejecutan los bloques de código correspondientes.

```
switch(variable)
{
  case valor_1:
    bloque de código1;
    break;

  case valor_2:
    bloque de código2;
    break;

  ...

  case valor_n:
    bloque de código;
    break;

  default:
    bloque de código;
    break;
}
```

La sentencia **switch** presenta las siguientes características:

- ⇒ Las instrucciones que forman parte del `switch` se encierran entre las llaves `{ y }`.
- ⇒ Dentro del `switch` se definen todas las comparaciones que se quieren realizar sobre el valor de la variable. Cada comparación se indica mediante la palabra reservada **case** seguida del valor con el que se realiza la comparación. Si el valor de la variable utilizada por `switch` coincide con el valor indicado por `case`, se ejecutan el **bloque de código** dentro de ese `case`.
- ⇒ Los **bloques de código** dentro del **case** no necesitan ir encerrados entre llaves.
- ⇒ Las comparaciones se realizan por orden, desde el primer `case` hasta el último, por lo que es muy importante el orden en el que se definen los `case`.
- ⇒ Si ningún valor de la variable del `switch` coincide con los valores definidos en los `case` se utiliza el valor **default**. Este bloque corresponde al último **else** de la sentencia **if-else**.
- ⇒ Los *bloques de código* se ejecutan o no dependiendo del valor de una variable, no de una condición.
- ⇒ Los valores **case** han de ser expresiones constantes del mismo tipo que la `variable`.
- ⇒ La palabra clave **break** sirve para finalizar la ejecución de **switch** después de haberse ejecutado un bloque.
- ⇒ El valor de la variable ha de ser de tipo entero o carácter (cerrado entre comillas).
- ⇒ Si ha de ejecutarse un bloque cuando el resultado de evaluar la variable pertenezca a un **conjunto de valores** (del 1 al 7, vocales minúsculas, horas...), éstos han de enumerarse (cada uno precedido de la palabra clave **case** y seguido de un signo de dos puntos):



```

switch (c)
{
    case '1': case '3': case '5': case '7': case '9':
        alert("c es un número impar");
        break;
    case '0': case '2': case '4': case '6': case '8':
        alert("c es un número par");
        break;
    case ' ':
        alert("c es un espacio");
        break;
    default :
        alert("c no es un número ni un espacio");
}

```

**Ejemplo 4:** Vamos a realizar un programa que nos da la 'nota' cualitativa a partir de la cuantitativa, es decir, el programa nos pide el número total de preguntas y el número de respuestas acertadas, y tendrá que devolver la 'nota' cualitativa.

```

<HTML>
<SCRIPT LANGUAGE='JavaScript'>
// NOTA CUALITATIVA.HTM
var num,bien,notanum,notacual;
num=parseInt(prompt('Escribe el número total de preguntas',''));
bien=parseInt(prompt('Escribe el número de respuestas acertadas',''));
notanum=parseInt(10*bien/num);
switch(notanum)
{
    case 0:
        notacual='Muy Deficiente';
        break;
    case 1:
        notacual='Muy Deficiente';
        break;
    case 2:
        notacual='Deficiente';
        break;
    case 3:
        notacual='Deficiente';
        break;
    case 4:
        notacual='Insuficiente';
        break;
    case 5:
        notacual='Suficiente';
        break;
    case 6:
        notacual='Bien';
        break;
    case 7:
        notacual='Notable';
        break;
    case 8:
        notacual='Notable';
        break;
}

```

```

        case 9:
            notacual='Excelente';
            break;
        case 10:
            notacual='Matricula de Honor';
            break;
    }
    alert('La nota cualitativa es '+notacual);
</SCRIPT>
</HTML>

```

Cuando lo que queremos expresar es un rango lo hacemos definiendo varias instrucciones case seguidas:

**Ejemplo:** Leer una nota y escribir en pantalla la calificación:

```

Switch (nota)
{
    Case 0: case 1: case 2: case 3: case 4:
        alert("suspendido");
        break;
    case 5:
        alert("suficiente");
        break;
    case 6:
        alert("bien");
        break;
    case 7: case 8:
        alert("notable");
        break;
    case 9: case 10:
        alert("sobresaliente");
        break;
}

```



### 6.2.2 ESTRUCTURAS REPETITIVAS

Es aquella que contiene conjunto de instrucciones que se **repiten** un número finito de veces. A esta repetición (o iteración) se le llama bucle. Todo bucle tiene que llevar asociada una **condición**, que es la que va a determinar si se siguen ejecutando las instrucciones o si se sale del bucle.

Existen dos tipos de estructuras repetitivas:

- 1º). ESTRUCTURA for:** Se conoce el número de veces que un conjunto de instrucciones se van a ejecutar (20, 5, 2 veces).

**Ejemplo:** Introduce 100 números y escríbelos en pantalla

- 2º).** En la que el número de repeticiones es desconocido y se realizará hasta que la condición se evalúe como FALSA.

**Ejemplo:** Introduce números y escríbelos en pantalla hasta que se introduzca un cero

- a). ESTRUCTURA while:** Si la verificación de la condición se hace al inicio, de forma que:

- Si la condición es VERDADERA, se ejecutan las instrucciones del bucle y después de la última se vuelve al principio a preguntar por la condición.
- Al evaluarse la condición antes de entrar en el bucle, si la condición al ser evaluada la primera vez es falsa, no entraremos nunca en el bucle. Por tanto, el bucle puede que se ejecute 0 veces.

- b). ESTRUCTURA do...while:** Si la verificación de la condición se hace al final. Escogeremos este tipo de bucle cuando queramos que como mínimo entre una vez y ejecute todas las instrucciones que se encuentran dentro del bucle.

### 6.2.2.1 ESTRUCTURA WHILE

Para el caso en que no se sepa el nº de veces que se ejecutará el bucle. Su sintaxis es:

```
while (condicion)
{
    ...
}
```

El funcionamiento del bucle es como sigue:

- Si la condición no se cumple ni siquiera la primera vez, el bucle no se ejecuta.
- Si la condición se cumple (se evalúa como VERDADERA), entrará en el bucle y se ejecutan las instrucciones una vez. Se vuelve a comprobar la condición, y si se sigue cumpliendo la condición, se vuelve a ejecutar el bucle y así se continúa hasta que la condición deje de cumplirse, es decir, se evalúe como FALSA.

Las variables que controlan la condición deben modificarse dentro del propio bucle, ya que de otra forma, la condición se cumpliría siempre y el bucle se repetiría indefinidamente. A esto se le denomina **Bucle infinito**.

**Ejemplo1:** Programa que introduces números y los escribes en pantalla, hasta que introduzcas un número negativo.

```
<SCRIPT LANGUAGE='JavaScript'>
var num;
num=parseInt(prompt("Introduce un número:", ""));
while (num>0)
{
    document.write (num+"<br>");
    num=parseInt(prompt("Introduce un número:", ""));
    /*hay que volver a pedir otro número porque sino
    sería siempre el mismo*/
}
</SCRIPT>
```

**Ejemplo1:** Programa que escriba tu nombre tantas veces como se le indique.

#### 1º). ANÁLISIS

```
EScribr tu nombre:
ISABEL
¿Cuántos veces quieres escribir tu nombre?
4

ISABEL
ISABEL
ISABEL
ISABEL
```

#### 2º). DISEÑO

```

<SCRIPT LANGUAGE='JavaScript'>
// nombre.HTM
var nom;
var indice=1;
var num;
var respuesta=' ';
nom=prompt('Escribe tu nombre','');
num=parseInt(prompt('Cuántas veces quieres que lo repita',''));
while (indice <= num)
{
    document.write(nom+"<br>");
    indice++;
}
</SCRIPT>

```

**Ejemplo 1:** modifica el algoritmo de la suma de dos números de forma que sume n números que introduciremos por teclado, siendo n el número de números que queremos sumar.

### 3º). ANÁLISIS

¿Cuántos números deseas sumar?  
4

Introduce un número: 2  
Introduce un número: 10  
Introduce un número: 3  
Introduce un número: 5  
LA SUMA ES: 20

Tenemos que introducir tantos números como se indica e ir sumándolos uno a uno. Por último mostramos el resultado de dicha suma

### 4º). DISEÑO

```

<HTML>
<SCRIPT LANGUAGE='JavaScript'>
// SUMA n POSITIVOS_B.HTM
var suma=0;
var cont=0;
var n;
var num;
n=parseInt(prompt("¿Cuántos números deseas sumar:",""));
while (cont<=n)
{
    num=parseInt(prompt("Introduce un número:",""));
    suma=suma+num;
    cont++;
}
alert ("La suma es: "+suma);
</SCRIPT>
</HTML>

```

**5º). PRUEBA**

	n	num	suma	cont	SALIDA EN PANTALLA
INICIALMENTE	4		0	0	
DENTRO DEL BUCLE		2	0+2	0+1	
		10	0+2+10	1+1=2	
		3	0+2+10+3	2+1=3	
		5	0+2+10+3+5	3+1=4	LA SUMA ES 20

**Ejemplo2:** Algoritmo que calcule el producto de varios números introducidos por teclado. En el momento que introduzcamos un número negativo deberá mostrar el resultado en pantalla.

**1º). ANÁLISIS**

Introduce un número: 2  
 Introduce un número: 10  
 Introduce un número: 3  
 Introduce un número: -5  
 EL PRODUCTO ES: 60

**2º). DISEÑO**

```
<HTML>
<SCRIPT LANGUAGE='JavaScript'>
// PRODUCTO n POSITIVOS_a.HTM
var producto=1; // Inicializamos a 1 porque forma parte de una expresión que será
una multiplicación
var num;
num=parseInt(prompt("Introduce un número:",""));
while (num>0)
{
    producto=producto*num;
    num=parseInt(prompt("Introduce un número:",""));
}
alert ("el producto es: "+producto);
</SCRIPT>
</HTML>
```

**3º). PRUEBA**

	P	num	SALIDA EN PANTALLA
INICIALMENTE	1	2	
DENTRO DEL BUCLE	1*2	10	
	1*2*10	3	
	1*2*10*3	-5	EL RESULTADO ES: 60

### 6.2.2.2 ESTRUCTURA DO...WHILE

En esta estructura siempre se ejecutan las instrucciones del bucle al menos la primera vez y se comprobará la condición de finalización al final del bucle

```
do
{
    ...
}
while(condicion)
```

**Ejemplo1:** Introduce números hasta que introduzcas un 5 (mientras sea distinto de 5 se considerará erróneo).

```
<SCRIPT LANGUAGE='JavaScript'>
var num;
do //esta instrucción no impide entrar en el bucle
{
    num=parseInt(prompt("Introduce un número:", ""));
}
while (num<>5)
document.write ("El numero correcto es: "+ num);
</SCRIPT>
```

**Ejemplo2:** Modifica el algoritmo de la suma de n números utilizando la estructura do ..while

```
<HTML>
<SCRIPT LANGUAGE='JavaScript'>
// SUMA n POSITIVOS_a.HTM
var suma=0;
var cont=0;
var n;
var num;
n=parseInt(prompt("¿Cuántos números deseas sumar:",""));
do
{
    num=parseInt(prompt("Introduce un número:",""));
    suma=suma+num;
    cont++;
}
while (cont<n)
alert ("La suma es: "+suma);
</SCRIPT>
</HTML>
```

**Pero, ¿Qué inconveniente presenta este último algoritmo? Supongamos que no queremos introducir ningún número, con lo que  $n=0$ . Pero sin embargo, entra en el bucle, ejecuta la suma e incrementa el contador en 1. Con lo cual sería un bucle infinito, pues la variable *cont* nunca será igual a *n*.**

### 6.2.2.3 FOR

Se utiliza cuando se sabe ya antes de ejecutar el bucle el número exacto de veces que hay que ejecutarlo.

Para ello el bucle llevara asociado una variable que denominamos **variable índice**, a la que le asignamos un valor inicial (**inicialización**) y determinamos cual va a ser su valor final (**condición finalización**). Esta variable será la que cuente el nº veces que se ejecuta el bucle, por tanto, en cada pasada del bucle deberá incrementarse, o decrementarse, dependiendo el caso (**actualización**)

```
for(inicializacion; condicion; actualizacion)
{
    ...
}
```

**Ejemplo1:** Realiza un bucle que pida tu nombre y lo escriba en el navegador 10 veces

```
<SCRIPT LANGUAGE='JavaScript'>
var nombre,i;
nombre=prompt("Introduce tu nombre:", "");
for (i=1; i<=10; i++)
{
    document.write (nombre+"<br>");
}
</SCRIPT>
```

Inicialización  
(Empieza en 1)

Condición de finalización  
(Finalizará cuando la i sea mayor de 10)

Actualización  
(La i se incrementa en 1 en cada pasada del bucle)

**Ejemplo1:** Resuelve el algoritmo de la suma de n números utilizando esta estructura

```
<HTML>
<SCRIPT LANGUAGE='JavaScript'>
// SUMA n POSITIVOS.HTM
var suma=0;
var n;
var num;
n=parseInt(prompt("¿Cuántos números deseas sumar:",""));
for(var i=1;i<=n;i++)
{
    num=parseInt(prompt("Introduce un número:",""));
    suma=suma+num;
}
alert ("La suma es: "+suma);
</SCRIPT>
</HTML>
```

**Ejemplo2:** Mostrar en pantalla todos los números pares hasta el 300

```
<HTML>
<SCRIPT LANGUAGE='JavaScript'>
// pares hasta 300.HTM
for(i=2; i<=300;i=i+2)
```

```
        alert (i);  
</SCRIPT>  
</HTML>
```

**Ejemplo:** Si deseamos mostrar los impares el algoritmo es el siguiente:

```
<HTML>  
<SCRIPT LANGUAGE='JavaScript'>  
// impares hasta 300.HTM  
for(i=1; i<=300;i=i+2)  
    alert (i);  
</SCRIPT>  
</HTML>
```

## 7 FUNCIONES

Las funciones es una de las partes fundamentales del código JavaScript. Se pueden definir como un conjunto de instrucciones que realiza una determinada acción o calculan un valor, y que va a ser llamada o usada en distintas partes del código.

Cuando se diseña un programa complejo, se suele utilizar un mismo conjunto de instrucciones varias veces a lo largo de todo el código, como por **ejemplo** el cálculo de un precio total a la hora de diseñar una tienda on\_line.

Esto presenta varios inconvenientes:

- 1º). El código fuente es más largo y por tanto enlentecemos la ejecución del programa
- 2º). Modificar una instrucción de ese conjunto requiere la modificación de esa misma instrucción a lo largo del programa, lo cual es trabajoso y propenso a cometer errores.

Para solucionar esto diseñamos las funciones.

Una **función** es un trozo de código o conjunto de instrucciones a las que se les da un nombre y que se puede reutilizar varias veces haciendo una llamada a la función. A esto se le denomina **invocar** una función.

Ejemplo: funciones que ya vienen predefinidas en JavaScript y que utilizamos muy a menudo, `parseInt()` y `prompt()`.

Por tanto, para diseñar un programa complejo, que empieza a alcanzar cierto tamaño, la única forma de estructurarlo es dividirlo en secciones, es decir, en unidades más pequeñas (funciones). El programa irá llamando (invocando) a las funciones a medida que avanza la ejecución.

En este sentido:

- En vez de teclear esa secuencia de instrucciones varias veces se diseña la función y se invoca (se hace una llamada) a la función tantas veces sea necesaria a lo largo del programa.
- Permiten simplificar el código y estructurarlo con objeto de hacerlo más fácil de entender y de modificar.

El concepto y el funcionamiento es el mismo que por ejemplo, en Excel. Una función se caracteriza por un nombre y entre paréntesis los argumentos (en caso de tenerlos), en concreto en JavaScript, separados por comas:

Ejemplo:

EXCEL	JAVASCRIPT
<b>SUMA(A1:A10;F1:F10)</b>	SUMA(a,b)

Además, hay funciones que ya vienen diseñadas en JavaScript, como por ejemplo, `parseInt()`, y otras que las diseñaremos nosotros. Por tanto, a la hora del diseño y utilización de las funciones deberemos tener en cuenta los siguientes pasos:

- PASO 1.** Declaración de la función
- PASO 2.** Llamada o invocación de la función
- PASO 3.** Devolución o retorno de valor/es de la función



## 7.1 DECLARACIÓN DE UNA FUNCIÓN

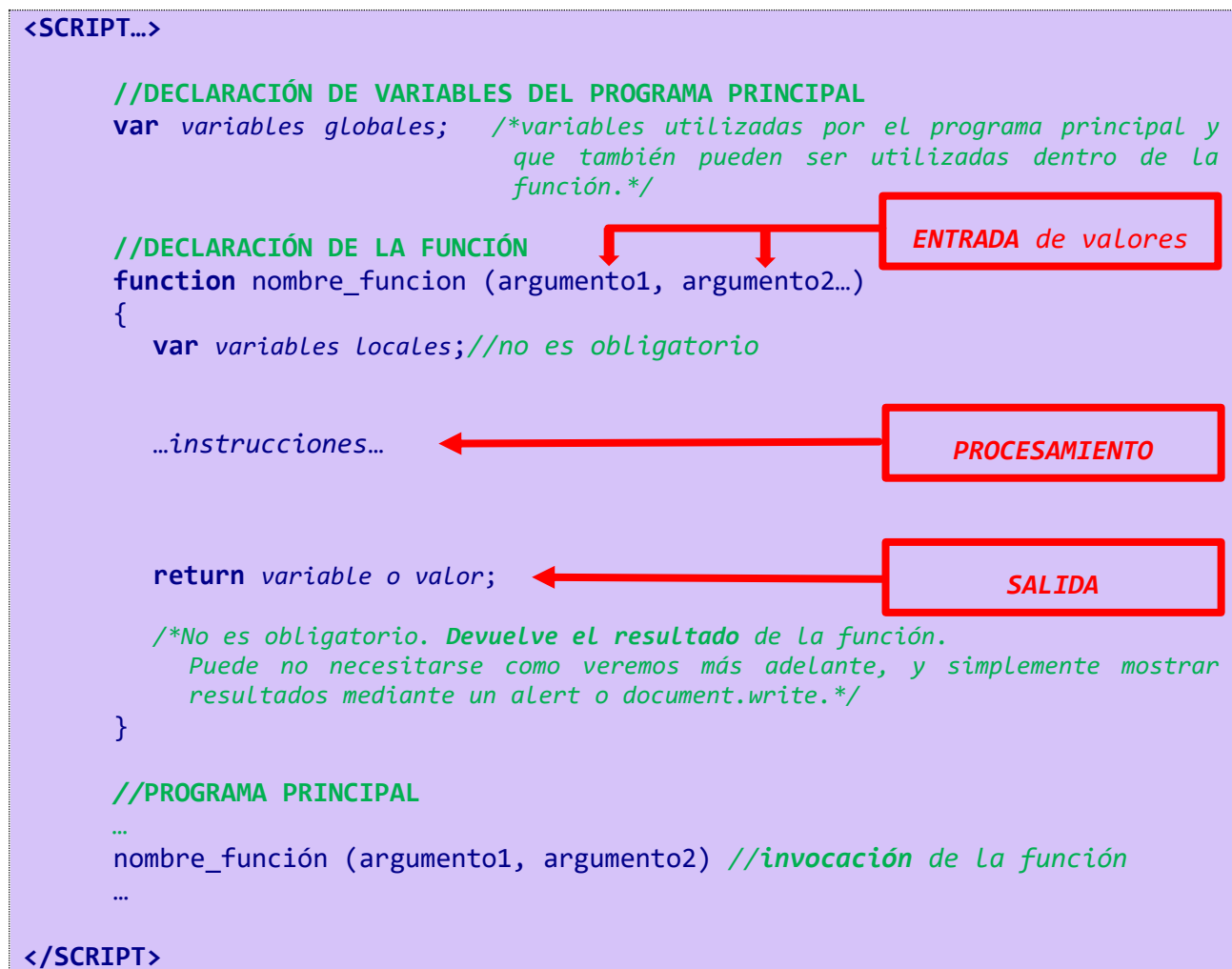
Antes de comenzar con el diseño de la función hay que **declarar** la función, al igual que declaramos las variables. A la hora del diseño y utilización de la función tenemos que diferenciar entre la declaración de la función y la invocación (llamada) a la función

La **declaración de funciones** se hace a continuación de la *declaración de variables* (al inicio del código) del programa principal como mostraremos a continuación.

Además, sabemos que una **función** es un pequeño programa que forma parte del programa principal y que realiza unas instrucciones de manera independiente al resto del programa. Por tanto:

- 1º) tiene unos valores de ENTRADA, llamados **Parámetros o Argumentos**.
- 2º) un PROCESAMIENTO de esos valores, mediante las **instrucciones**
- 3º) unos valores de SALIDA, es decir el resultado de la función, que lo devuelve, o bien las **instrucciones de escritura** alert o document.write o mediante la instrucción **return**

**Sintaxis declaración de la función:**



**Ejemplo:** Diseña una función para sumar 2 números que se proporcionaran como entrada. (No hace falta retornar un valor. Solo mostrarlo en pantalla)

```
<SCRIPT LANGUAGE="JavaScript"> |
var a,b; //variables globales
function sumar(a,b)
{
    var suma;//variables locales
    suma = a + b;
    alert("la suma de: "+a+"+"+b+"="+suma);
}
sumar(1,2); //invocación de la función
...        //resto de líneas de código
sumar(3,5); //otra vez invocación de la función
</SCRIPT>
```

## 7.2 ÁMBITO DE LA FUNCIÓN: VARIABLES LOCALES Y GLOBALES

Las variables pueden definirse dentro o fuera de la función. Si se define una variable dentro de una función, estas variables no son accesibles desde fuera de la función. Se distingue pues:

- **Variables globales:** pueden utilizarse en cualquier parte del código (incluidas las funciones) y son declaradas fuera de toda función con la instrucción **var**.
- **Variables locales:** se definen con la instrucción **var** dentro de una función y sólo pueden ser utilizadas dentro de ésta.

**Ejemplo:** suma variables locales y globales

```
<SCRIPT LANGUAGE="JavaScript">
var v_global=10;
function suma()
{
    var v_local1=20;
    var v_local2=v_local1+v_global;
    alert ("La suma de la variable local y la global es "+ v_local2);
}
suma();
alert ("La variable global es "+v_global);
</SCRIPT>
```

Si en el interior de una función, las variables se declaran mediante **var** se consideran locales y las variables que no se han declarado mediante **var** en el interior de la función, se convierten automáticamente en variables globales y pueden ser utilizadas fuera de la función.

En el caso de coincidir en nombre las variables locales y globales, las variables locales prevalecen sobre las globales, pero sólo dentro de la función.

### 7.3 ENTRADA DE VALORES: ARGUMENTOS O PARÁMETROS

Los valores de entrada que necesitan las funciones se llaman **argumentos** y vienen a continuación del nombre de la función, entre paréntesis y separados por comas. Hay funciones que no necesitan estos valores de entrada

Los valores utilizados en los argumentos pueden ser de todo tipo: números, cadenas de caracteres...

### 7.4 LLAMADA O INVOCACIÓN A LA FUNCIÓN

Una vez diseñada la función (recordemos que la función se declara detrás de la declaración de variables) para ser utilizada hay que llamar a la función, es decir, **invocarla**, desde el programa principal, es decir, desde la parte de código donde se necesite que esté por debajo de la declaración de variables y funciones.

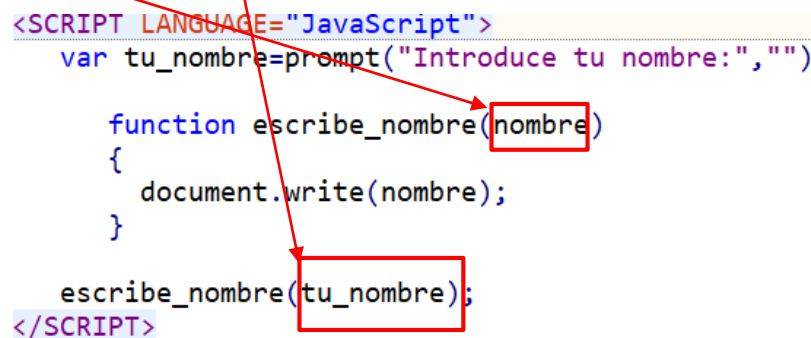
Para invocar la función basta con escribir su nombre, y entre paréntesis y separados por comas, los argumentos o parámetros (en caso de necesitarlos), es decir, los valores de entrada a la función. De esta forma, se ejecutarán todas las instrucciones en ella contenida.

**Sintaxis llamada a la función:**

```
nombrefuncion (argumento1, argumento2...);
nombrefuncion ();
```

Al invocar la función, se deben incluir los valores que se le van a pasar a la función como se muestra en el ejemplo de la suma de 2 números.

El argumento de la función y la variable con que es invocada no tienen por qué tener el mismo nombre:



```
<SCRIPT LANGUAGE="JavaScript">
  var tu_nombre=prompt("Introduce tu nombre:","")

  function escribe_nombre(nombre)
  {
    document.write(nombre);
  }

  escribe_nombre(tu_nombre);
</SCRIPT>
```

Red arrows in the original image point from the text 'El argumento de la función y la variable con que es invocada no tienen por qué tener el mismo nombre:' to the parameter 'nombre' in the function definition and the variable 'tu\_nombre' in the function call.

La invocación a la función puede ser realizada de diferentes formas como se muestra en el siguiente ejemplo, pero siempre dependerá de la forma en que se devuelva el valor de la función como veremos más adelante.

**Ejemplo:** Suma de 2 números:

```
var total = sumar(3,5); // da como resultado 8
...
alert(sumar(7,3)); // da como resultado 10
...
document.write(total + sumar(10,-3)); // al total anterior le vuelve a sumar 2 nuevos valores
```

## 7.5 RETURN: RETORNO DE UN VALOR: VALOR DE SALIDA.

Las funciones no solamente pueden recibir variables y datos, sino que también pueden devolver los valores que han calculado.

Para devolver valores, o bien se realiza mediante las **instrucciones de escritura** (document.write o alert), o bien, devolviendo un valor mediante **instrucción de retorno**:

```
return nombre_variable
```

La sentencia return **finaliza la ejecución de la función** y devuelve un valor (numérico, texto, booleano, array,...) al lugar del programa principal donde sea invocada a la función.

```
function nombre_funcion (argumento1, argumento2...)
{
    ...instrucciones...
    return valor; //si queremos que la función devuelva algún valor
}
```

No es preciso que una función devuelva nada. No es necesario usar *return*.

Cuando las funciones devuelven un valor, ese valor se puede asignar o almacenar en otra variable en el momento que se invoca (llama) a la función desde el programa principal:

```
var resultado = nombre_función(parámetros);
```

```
<SCRIPT LANGUAGE="JavaScript">
var total1, total2;
function sumar(a,b)
{
    var suma = a + b;
    //alert("la suma de: "+a+"+"+b+"="+suma);
    return suma;
}
total1=sumar(1,2);
alert("la suma de 1+2= "+total1);
total2=sumar(3,5);
alert("la suma de 3+5= "+total2);
</SCRIPT>
```

También se puede mostrar el resultado directamente sin necesidad de almacenarlo:

```
<script language=javascript>
//Usar la función para mostrar "Hola" seguido del nombre de la persona que le pasaremos como parámetro
var nombre;

function saludo(nombre)
{
    return "Hola " + nombre;
}

nombre= prompt("Introduce tu nombre","");
document.write(saludo(nombre));
</script>
```

Aunque las funciones pueden devolver valores de cualquier tipo (números, texto, valores booleanos ...), **solamente pueden devolver un valor cada vez que se ejecutan** y por tanto, solo puede ejecutarse una instrucción return. Entonces, ¿Qué ocurre si necesitamos que se devuelva más de un valor?. La solución sería utilizar una estructura de datos que no es una variable sino que alberga un conjunto de variables. Esta estructura se denomina **array** y la veremos en el apartado siguiente.

## 7.6 UTILIZACIÓN DE FUNCIONES PARA VALIDAR VARIABLES

Una función puede devolver un valor booleano (1 o 0, true o false respectivamente) que se utiliza para validar variables: edad correcta, fecha correcta, que se rellene un campo obligatorio, que se seleccione el valor de una lista desplegable, email correcto, que se haya introducido un número donde así se requiere, que el nº introducido sea correcto,...etc.

**Ejemplo:** la siguiente función comprueba si el nº es positivo. Si lo es, devuelve un 1 y si no, devuelve un 0.

```
function numero_positivo(n)
{
    if(n<0)
    {
        return 0;
    }
    return 1;
}
```

Como la sentencia **return** finaliza la ejecución de la función, fíjate que esta instrucción no está dentro de un else, ya que si el número es negativo finalizaría la función en la instrucción **return 0**. En caso de que sea positivo, no entraría en el if y ejecutaría automáticamente la instrucción **return 1**

Como se ha indicado antes, cuando una instrucción de retorno se llama en una función, se detiene la ejecución de la función. **Quiere esto decir, que si pusiéramos instrucciones detrás de la instrucción return, no se ejecutarían, se ignoran.** Es por ello, que la instrucción *return* suele ser la última instrucción de una función.

**Ejemplo:** supongamos que queremos diseñar la siguiente función para comprobar si un nº es positivo y devolvemos el mensaje de error en caso de que no lo sea:

```
<script language="Javascript">
    var n // Declaración de variables

    function numero_positivo(n) // Declaración de la función
    {
        if(n<0)
        {
            return 0;
            alert("hola")
        }
        return 1;
    }

    // Programa principal
    n=parseInt(prompt("Introduce un nº:", ""));
    if (!numero_positivo(n))
    {
        alert("ERROR. EL Nº INTRODUCIDO ES NEGATIVO")
    }
    else
    {
        alert("CORRECTO")
    }
}
</SCRIPT>
```

Hemos dicho que detrás de una instrucción **return** no se ejecutaría nada. Por tanto la instrucción `alert("hola")` dentro del if y la instrucción `"return 1"` fuera de ella, no se ejecutarían en el caso de introducir un número negativo

Significa que si el nº no es positivo devolverá un mensaje de error

## 7.7 ASPECTOS A TENER EN CUENTA

- Dentro de una función se pueden utilizar todos los elementos de programación que hemos ido estudiando: variables, arrays, estructuras condicionales, bucles, etc.
- Las funciones se pueden anidar y pueden llamar también a otra función. Pero hay que minimizar las interdependencias entre las funciones.
- Cada función debe resolver un único problema. Si la función empieza a crecer, se debe dividir dicha función en unidades más pequeñas.
- Para el nombre de las funciones hay que elegir siempre nombres sencillos y que sean representativos de la funcionalidad de la función.
- El orden de los argumentos es importante, ya que el primer dato que se indica en la llamada, será el primer valor que espera la función... y así sucesivamente.
- Se puede utilizar un número ilimitado de argumentos, aunque si su número es muy grande, se complica en exceso la llamada a la función.
- No es obligatorio que coincida el nombre de los argumentos que utiliza la función y el nombre de los argumentos que se le pasan.
- En el momento que se ejecute una instrucción `return` la función finaliza

## 8 ESTRUCTURA DE DATOS ARRAYS

Como en la mayoría de los lenguajes de programación, en JavaScript podemos trabajar con arrays (arreglos, vectores o matrices). La importancia de esta **estructura de datos** radica en que permiten organizar una colección de datos que comparten el mismo nombre.

Un array es una colección de variables, que pueden ser todas del mismo tipo o cada una de un tipo diferente. Sirven para almacenar valores que tienen una relación entre sí.

**Ejemplo 1:** si queremos utilizar en un programa los días de la semana, con lo que sabemos hasta ahora se podrían crear siete variables de tipo texto:

```
var dia_sem1 = "Lunes";

var dia_sem2 = "Martes";

...

var dia_sem7 = "Domingo";
```

¿Pero qué pasaría si se tuviera que guardar el nombre de los meses del año, el nombre de todos los países del mundo?

La solución es agrupar todas esas variables relacionadas en una colección de variables de la siguiente forma:

```
var dia_sem = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado", "Domingo"];
```

### 8.1 DECLARACIÓN DEL ARRAY

La declaración de un array se hace de misma forma que se declara cualquier variable:

```
var nombre_array;
```

Si recordamos definíamos una variable como una posición de memoria (cuadradito) a la que le damos un nombre. Pues bien, ahora con los arrays no hablamos de una sola posición de memoria si no de un conjunto de cuadraditos a los que se les da un solo nombre, es una gran variable que no almacena solo un dato, sino un conjunto de datos relacionados. Gráficamente lo podemos representar de la siguiente manera:

**VARIABLE**



**ARRAY**



Para definir un array (inicializarlo), se utilizan los corchetes [ y ] con los elementos internos separados por comas:

```
var nombre_array = [valor1, valor2, ..., valorN];
```

**Ejemplo:** var dia\_sem=["Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado", "Domingo"];

Otra forma de declarar el array es:

```
var nombre_array=new Array(valor1, valor2,..., valorN);
```

**Ejemplo :**var dia\_sem=new Array("Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado", "Domingo");

Pero hay que llevar cuidado con esta última forma de declararlo porque si estamos manejando valores de tipo numérico y ponemos:

- `var número=new Array(10,11)` sabemos que estamos creando un array de 2 posiciones, pero:
- `var número=new Array(10)` significa que estamos creando un array de 10 posiciones (casillas)

También se puede declarar un array vacío (array indeterminado) ya que puede interesar rellenarlo más tarde:

```
Var nombre=[]
```

```
Var nombre=new Array()
```

## 8.2 ACCESO A LOS ELEMENTOS DEL ARRAY

Para acceder a cada uno de los elementos hay que indicar su posición (o también denominado **índice** representado por la **i**) dentro del array empezando en la **posición 0**. Para ello, se indica el nombre del array y entre corchetes la posición del elemento dentro del array:

```
document.write(dia_sem[0]) escribirá "Lunes"
```

```
document.write(dia_sem[6]) escribirá "Domingo"
```

Gráficamente quedaría de la siguiente forma:

i	0	1	2	3	4	5	6
dia_sem	Lunes	Martes	Miércoles	Jueves	Viernes	Sábado	Domingo

De esta forma si queremos escribir en pantalla la posición 3 del array escribiríamos:

```
document.write(dia_sem[3])
```

Mostrando como resultado en la pantalla del navegador:

Jueves

**Ejemplo 1:** Mostrar en pantalla todos los elementos del array utilizando el bucle for:

### PROGRAMA

```
<SCRIPT LANGUAGE='JavaScript'>
  var dia_sem=["Lunes", "Martes", "Miércoles", "Jueves",
  "Viernes", "Sábado", "Domingo"];
  //Escritura del contenido del array separado por comas:
  document.write(dia_sem);
  //Escritura del contenido del array mediante un bucle for
  for(var i=0;i<=6;i++)
  {
    document.write(dia_sem[i]+"<br>")
  }
</SCRIPT>
```

### SALIDA EN PANTALLA

Lunes, Martes, Miércoles, Jueves, Viernes,  
Sábado, Domingo

Lunes  
Martes  
Miércoles  
Jueves  
Viernes  
Sábado  
Domingo



### 8.3 PROPIEDADES Y MÉTODOS. HERRAMIENTAS PARA MANIPULAR EL ARRAY

Existe una serie de herramientas para poder manipular el array para calcular la longitud del array, agregar elementos, eliminarlos, ordenarlos, unir dos arrays,...etc. Estas herramientas se denominan PROPIEDADES y MÉTODOS.

- Las **propiedades** son las características que posee el array, como su longitud.
- Los **métodos** son las acciones que se pueden realizar: agregar, eliminar, ordenar,...

Tanto en las propiedades como en los métodos, para hacer referencia ellos, se utiliza la metodología del punto:

```
nombre_array.nombre_propiedad
o
nombre_array.nombre_metodo(parametros)
```

#### 8.3.1 PROPIEDAD LENGTH

Nos indica la longitud o número de elementos que contiene el array:

**Ejemplo2:** Vamos a mostrar la longitud del array y la vamos a utilizar para mostrar sus elementos en en una columna.

PROGRAMA	SALIDA EN PANTALLA
<pre>&lt;SCRIPT LANGUAGE='JavaScript'&gt;   // Escribir los elementos de un array   var dia_sem= new Array("Lunes", "Martes", "Miércoles", "Jueves",     "Viernes", "Sábado", "Domingo");   document.write("La longitud de del array es:"+dia_sem.length+"&lt;br&gt;");   for(var i=0;i&lt;=dia_sem.length-1;i++)   {     document.write(dia_sem[i]+"&lt;br&gt;")   } &lt;/SCRIPT&gt;</pre>	<pre>Lunes Martes Miércoles Jueves Viernes Sábado Domingo</pre>

#### 8.3.2 MÉTODO PUSH

**Agregar** elementos **al final** del array y devuelve la nueva longitud del array.

```
Nom_array.push(elemento1, elemento2,...)
```

**Ejemplo3:** Vamos a agregar la palabra "Adios" al final del array

PROGRAMA	SALIDA EN PANTALLA
<pre>&lt;SCRIPT LANGUAGE='JavaScript'&gt;   var dia_sem= new Array("Lunes", "Martes", "Miércoles", "Jueves",     "Viernes", "Sábado", "Domingo");   //agregamos un elemento al final del array   dia_sem.push("Adiós")   document.write("la nueva longitud es:"+ dia_sem.push("Adiós"))   for(var i=0;i&lt;=dia_sem.length-1;i++)   {     document.write(dia_sem[i]+"&lt;br&gt;")   } &lt;/SCRIPT&gt;</pre>	<pre>La nueva longitud es 8 Lunes Martes Miércoles Jueves Viernes Sábado Domingo Adiós</pre>

### 8.3.3 MÉTODO UNSHIFT

**Agregar** elementos **al principio** del array y devuelve la nueva longitud del array.

```
Nom_array.unshift(elemento1, elemento2,...)
```

**Ejemplo4:** Al anterior programa le vamos a agregar la palabra "Hola" al principio del array

#### PROGRAMA

```
<SCRIPT LANGUAGE='JavaScript'>
  var dia_sem= new Array("Lunes", "Martes", "Miércoles", "Jueves",
    "Viernes", "Sábado", "Domingo");
  document.write("La longitud de del array es:"+dia_sem.length+"<br>");
  //agregamos un elemento al principio del array
  dia_sem.unshift("Hola")
  for(var i=0;i<=dia_sem.length-1;i++)
  {
    document.write(dia_sem[i]+"<br>")
  }
  document.write ("la nueva longitud es:"+ dia_sem.unshift("Hola"))

</SCRIPT>
```

#### SALIDA EN PANTALLA

**Hola**  
 Lunes  
 Martes  
 Miércoles  
 Jueves  
 Viernes  
 Sábado  
 Domingo  
 La nueva longitud es: 8

**Ejemplo5:** Vamos a agregar una palabra al array pero preguntando por la palabra a añadir

#### PROGRAMA

```
<SCRIPT LANGUAGE='JavaScript'>
  var dia_sem= new Array("Lunes", "Martes", "Miércoles", "Jueves",
    "Viernes", "Sábado", "Domingo");
  dia_sem.push("Adiós")
  //preguntamos por el elemento que deseamos agregar
  dia_sem.push(prompt("Introduce un elemento:"))
  for(var i=0;i<=dia_sem.length-1;i++)
  {
    document.write(dia_sem[i]+"<br>")
  }

</SCRIPT>
```

#### SALIDA EN PANTALLA

Lunes  
 Martes  
 Miércoles  
 Jueves  
 Viernes  
 Sábado  
 Domingo  
 Adios

Introduce un elemento:  
 Adios2

**Adios2**

### 8.3.4 MÉTODO POP

**Elimina** el **último** elemento del array y devuelve ese último elemento.

```
Nom_array.pop()
```

**Ejemplo6:** Vamos a eliminar la última palabra del array:

#### PROGRAMA

```
var dia_sem= new Array("Lunes", "Martes", "Miércoles", "Jueves", ...
//elimina ultimo elemento y lo devuelve
document.write("Elemento eliminado:"+ dia_sem.pop())
for(var i=0;i<=dia_sem.length-1;i++)
{
  document.write(dia_sem[i]+"<br>")
}
```

#### SALIDA EN PANTALLA

Lunes  
 Martes  
 Miércoles  
 ...  
**Adios**  
 Elemento eliminado: Adios

### 8.3.5 MÉTODO SHIFT

**Elimina el primer** elemento del array y devuelve dicho elemento. Este método cambia la longitud del array

Nom\_array.**shift()**

**Ejemplo6:** Vamos a eliminar la primera palabra del array:

#### PROGRAMA

```
<SCRIPT LANGUAGE='JavaScript'>
  var dia_sem= new Array("Lunes", "Martes", "Miércoles", "Jueves",
    "Viernes", "Sábado", "Domingo");
  dia_sem.unshift("hola")
  //elimina hola y almacena el valor en la variable eliminado
  var eliminado=dia_sem.shift()
  for(var i=0;i<=dia_sem.length-1;i++)
  {
    document.write(dia_sem[i]+"<br>")
  }
  document.write("Elemento eliminado:" + eliminado)
</SCRIPT>
```

#### SALIDA EN PANTALLA

**Hola**  
 Lunes  
 Martes  
 Miércoles  
 Jueves  
 Viernes  
 Sábado  
 Domingo  
 Elemento eliminado: Hola

### 8.3.6 MÉTODO CONCAT

Une dos o más arrays y devuelve un nuevo array.

Nom\_array1.**concat**(array2, array3,...)

**Ejemplo7:** Vamos a unir dos arrays

#### PROGRAMA

```
<SCRIPT LANGUAGE='JavaScript'>
  var dia_sem= new Array("Lunes", "Martes", "Miércoles", "Jueves", "Viernes",
    "Sábado", "Domingo");
  var otros=new Array ("y", "todos los meses")
  var menos=new Array("julio","agosto")
  var todos=dia_sem.concat(otros,menos);
  for(var i=0;i<=todos.length-1;i++)
  {
    document.write(todos[i]+"<br>")
  }
</SCRIPT>
```

#### SALIDA EN PANTALLA

Lunes  
 Martes  
 Miércoles  
 Jueves  
 Viernes  
 Sábado  
 Domingo  
 Y  
 Todos los meses  
 Julio  
 Agosto

### 8.3.7 METODO SPLICE

Elimina y/o añade nuevos elementos al array:

`Nom_array.splice(índice, cantidad, elemento1,...)`

**Índice:** posición en el array

**Cantidad:** es el número de elementos del array que se tienen que añadir o eliminar, de forma que:

- Si es igual a 0, se añadirá un elemento en la posición indicada en "índice" pero no se eliminará ningún elemento. Se debe especificar un nuevo elemento como mínimo,
- Si distinto de cero, se insertará el/los elementos en la posición indicada en "índice" eliminando los que si hubieran anteriormente.
- Si es mayor que el número de elementos restantes desde el inicio indicado, se eliminarán todos los elementos hasta el final del array.

**Elemento1, Elemento2,...:** Elementos que se desea añadir. Si se omiten simplemente se eliminan la cantidad de elementos desde la posición indicada

**Ejemplo9:** Elimina Sábado y Domingo y coloca en su lugar FIN DE SEMANA

#### PROGRAMA

```
<SCRIPT LANGUAGE='JavaScript'>
  var dia_sem= new Array("Lunes", "Martes", "Miércoles", "Jueves",
    "Viernes", "Sábado", "Domingo");
  //elimina Sábado y domingo y añade FIN DE SEMANA
  dia_sem.splice(5,2,"FIN DE","SEMANA")
  for(var i=0;i<=dia_sem.length-1;i++)
  {
    document.write(dia_sem[i]+"<br>")
  }
```

#### SALIDA EN PANTALLA

posición	contenido
0.	Lunes
1.	Martes
2.	Miércoles
3.	Jueves
4.	Viernes
5.	<b>FIN DE</b>
6.	<b>SEMANA</b>

**Ejemplo10** Añade FIN DE SEMANA antes de Sábado y Domingo

#### PROGRAMA

```
<SCRIPT LANGUAGE='JavaScript'>
  var dia_sem= new Array("Lunes", "Martes", "Miércoles", "Jueves",
    "Viernes", "Sábado", "Domingo");
  // añade FIN DE SEMANA: antes de Sábado y domingo
  dia_sem.splice(5,0,"FIN DE","SEMANA:")
  for(var i=0;i<=dia_sem.length-1;i++)
  {
    document.write(dia_sem[i]+"<br>")
  }
</SCRIPT>
```

#### SALIDA EN PANTALLA

posición	contenido
0.	Lunes
1.	Martes
2.	Miércoles
3.	Jueves
4.	Viernes
5.	FIN DE
6.	SEMANA
7.	<b>Sábado</b>
8.	<b>Domingo</b>

**Ejemplo11** Elimina Miércoles y Jueves**PROGRAMA**

```

<SCRIPT LANGUAGE='JavaScript'>
    var dia_sem= new Array("Lunes", "Martes", "Miércoles", "Jueves",
    "Viernes", "Sábado", "Domingo");
    // elimina Miercoles y Jueves
    dia_sem.splice(2,2)
    for(var i=0;i<=dia_sem.length-1;i++)
    {
        document.write(dia_sem[i]+"<br>")
    }
</SCRIPT>

```

**SALIDA EN PANTALLA**

posición	contenido
0.	Lunes
1.	Martes
2.	Viernes
3.	Sábado
4.	Domingo

**8.3.8 MÉTODO SLICE**

**Devuelve** los elementos seleccionados de una array sin incluir el elemento del num\_final. Pero ten en cuenta que **el array no se modifica**, este método solo devuelve los elementos solicitados que almacenaremos en otra variable:

Nom\_array.**slice**(num\_principio, num\_fin)

- **num\_principio**: posición inicial de la extracción. Si se omite es 0. Si es negativa extrae por el final del array tantos números negativos sean indicados
- **num\_fin**: posición final de la extracción sin incluir el elemento. Si se omite, extrae hasta el final. Si es negativa extrae por el final del array tantos números negativos sean indicados

**Ejemplo11** Dado el siguiente Array de números, extrae y muestra los 3 primeros elementos y a continuación los 2 últimos

**PROGRAMA**

```

<SCRIPT LANGUAGE='JavaScript'>
    var numeros = [3, 5, 7, 9];
    document.write(numeros+"<br/>");
    //Devuelve 2 elementos del interior del array
    var numeros2=numeros.slice(1,3)
    document.write(numeros2+"<br/>");
    document.write(numeros+"<br/>");
    // Extrae los 3 primeros elementos
    numeros2 = numeros. slice(0, -1);
    document.write(numeros2+"<br/>");
    //Extrae los 2 últimos
    numeros2= numeros. slice(-2);
    document.write(numeros2);
    numeros2= numeros. slice(-3,-1);
    document.write(numeros2);
</SCRIPT>

```

**SALIDA EN PANTALLA**

3,5,7,9

5,7

3,5,7,9

3,5,7

7,9

5,7

Pero si omitimos los 2 argumentos (num\_principio y un\_fin) el método slice se utilizaría para **copiar arrays**.

```
var copia= numeros.slice()
```

3,5,7,9

### 8.3.9 MÉTODO JOIN

**Une** y devuelve los elementos del array en una cadena unidos por el carácter que se indica en los argumentos:

```
Nom_array.join("carácter delimitador")
```

**Ejemplo8:** Vamos a separar los elementos mediante un "-"

PROGRAMA	SALIDA EN PANTALLA
<pre>&lt;SCRIPT LANGUAGE='JavaScript'&gt;   var dia_sem= new Array("Lunes", "Martes", "Miércoles", "Jueves",     "Viernes", "Sábado", "Domingo");   var otros=new Array ("y", "todos los meses")   var menos=new Array("julio","agosto")   var todos=dia_sem.concat(otros,menos);   //Agrupa los elementos únicamente y los escribe separados por   comas   alert(todos);   //Agrupa y Separa los elemento mediante "-"   alert(todos.join("-")); &lt;/SCRIPT&gt;</pre>	<pre>Lunes-Martes-Miercoles-...- -y-todos los meses-Julio- Agosto</pre>

### 8.3.10 METODO SORT Y REVERSE

Devuelve los elementos de un array ordenado.

```
Nom_array.sort()
```

- **sort()**: ordena de mayor a menor
- **reverse()**: ordena en orden inverso al array anteriormente ordenado

**Pero hay que tener en cuenta que ordenará los elementos tomándolos como si fueran caracteres ASCII de forma que si tenemos el siguientes array de números:**

```
<SCRIPT LANGUAGE='JavaScript'>
  var Num=[100, 1,21,2000,3, 23, 56, 98]
  document.write(Num.sort()+"<br>");
</SCRIPT>
```

Una vez aplicado el método **sort()** el resultado será:

1,100,2000,21,23,3,56,98

Si quisiéramos ordenarlos de forma ascendente tendríamos que pasarle como parámetro una función que de momento no es objeto de estudio.

[https://msdn.microsoft.com/es-es/library/4b4fbfhk\(v=vs.94\).aspx](https://msdn.microsoft.com/es-es/library/4b4fbfhk(v=vs.94).aspx)

## 9 PROGRAMACIÓN ORIENTADA A OBJETOS (POO)

La programación orientada a objetos (POO) es una nueva forma de programar que utiliza también los conceptos anteriormente estudiados pero que simplifica el código del programa. Realmente Javascript no es un lenguaje de programación orientado a objetos sino **basado en objetos**, es decir, lo que vamos a hacer es **usar objetos** que ya están creados en JavaScript o que crearemos nosotros.

Existen dos grandes grupos de lenguajes de programación:

- a. **PROGRAMACIÓN ORIENTADA A PROCEDIMIENTOS O FUNCIONES:** es la programación basada en las estructuras que hemos aprendido hasta ahora: estructuras de datos (variables, y de instrucciones, secuenciales y de control de flujo como son las condicionales (if...else) y los bucles (while, do...while, for) y las funciones.
- b. **PROGRAMACIÓN ORIENTADA A OBJETOS:** es la programación basada en ciertos objetos, que poseen unas propiedades y unas capacidades o funcionalidades, como por ejemplo, la estructura de datos ARRAY.

La PROGRAMACIÓN ORIENTADA A PROCEDIMIENTOS daba lugar a que nuestros programas pudieran llegar a ser muy extensos. Los **inconvenientes** con las que nos encontrábamos eran:

- 1º). Buscar un error era tremendamente complicado.
- 2º). Cuando una persona ajena al diseño de un programa en cuestión tenía que realizar algún cambio, resultaba muy complicado descifrar el código escrito por otra persona.
- 3º). El código resultante no podía ser reutilizado por otros programas parecidos

Para solucionar esto y simplificar la tarea de programación, de cara principalmente a la hora de corregir fallos o introducir nuevas funcionalidades en un programa, se creó la **PROGRAMACIÓN ORIENTADA A OBJETOS**. Se basa fundamentalmente, en dotar al código de programación el mismo comportamiento que tiene cualquier objeto del mundo real.

En este sentido, observando cómo funciona la VIDA REAL y se llegó a la conclusión de que:

- El mundo real está compuesto de **CLASES DE OBJETOS** (Ej: coches, mesas, sillas...)
- Todos los objetos comparten dos características:
  - Tienen un estado llamado **PROPIEDADES** (atributos o características) como el tamaño, color, peso, longitud,...( Ej: un coche tiene 4 ruedas)
  - Tienen un comportamiento llamado **CAPACIDADES** (métodos), es decir, lo que puede hacer el objeto (Ej: un coche arrancar, frenar, apagar, encender,...)
- Además, de cada objeto existen muchos **OBJETOS INDIVIDUALES** del mismo tipo (Ej: pueden haber muchos coches todos de la misma marca y modelo, que son fabricados de los mismos planos) .A cada objeto individual (Ej: coche FORD) se le denomina **INSTANCIA** de una **CLASE** de objetos (Ej: el coche FORD MONDEO es una instancia de la clase COCHES) Una **clase** es el plano del que se crean objetos individuales.

Todos estos conceptos se trasladaron al código de programación y se creó la PROGRAMACIÓN ORIENTADA A OBJETOS. Un ejemplo ya trabajado son los ARRAYS, con sus propiedades y métodos anteriormente citados.

**Ejemplo:** dia\_sem= new Array("lunes","martes", "miércoles","jueves","viernes","sábado","domingo")

El **OBJETO** dia\_sem es una **INSTANCIA** de la **CLASE** Array

Veámoslo mediante unos ejemplos del mundo real y como trasladarlo al código de programación.

	MUNDO REAL		PROGRAMACIÓN	
<b>CLASES</b> Moldes o plantillas a partir de los cuales se crean objetos	El mundo real está compuesto por: <ul style="list-style-type: none"> <li>• Coches (coche1, coche2,...)</li> <li>• Personas</li> <li>• Equipos de música</li> <li>• Ordenadores...</li> </ul>		<b>ARRAYS</b>	<b>BOTONES FORMULARIOS</b>
<b>OBJETOS</b> (es una entidad individual de una clase )	<ul style="list-style-type: none"> <li>• 1 coche</li> <li>• 1 persona</li> <li>• 1 equipo música</li> <li>• 1 ordenador</li> </ul>		<b>dia_sem=["Lunes","martes",..."Domingo"]</b> <b>dia_sem= new Array("Lunes",..."domingo")</b>	<b>BOTÓN para salir del FORMULARIO</b>
<b>PROPIEDADES</b> (Características). Permiten definir el estado del objeto	Los objetos tienen unas características:	<b>Length</b> (longitud de un Array)	<b>Botón activado o desactivado</b> <b>Style.width:</b> modificar el ancho de un botón	
	1 Coche: <ul style="list-style-type: none"> <li>• Color</li> <li>• Longitud</li> <li>• Ruedas...</li> </ul>	1 Persona: <ul style="list-style-type: none"> <li>• Nombre</li> <li>• Apellidos</li> <li>• Peso</li> <li>• edad</li> <li>• Altura</li> </ul>		
<b>MÉTODOS</b> (Capacidades o funcionalidades) Acciones que el objeto puede realizar por sí mismo	Los objetos tienen unas capacidades.		<b>Push</b> (añadir un elemento al final) <b>Pop</b> (Eliminar un elemento del final) <b>Concat</b> (concatenar 2 arrays)	<b>Clicar</b> un botón
	1 Coche: <ul style="list-style-type: none"> <li>• acelerar</li> <li>• frenar</li> <li>• girar...</li> </ul>	1 persona: <ul style="list-style-type: none"> <li>• comer</li> <li>• dormir</li> <li>• trabajar...</li> </ul>		



## 9.1 ¿QUÉ SON LAS CLASES?

Como hemos visto anteriormente una clase es un molde o plantilla a partir de los cuales se crean los OBJETOS (es decir, instanciamos un objeto). En este sentido se dice que las clases definen las características de los objetos.

Las clases se pueden agrupar en 3 categorías:

- **CLASES PREDEFINIDAS:** es decir, que ya están creadas en Javascript, incluyen las clases ARRAY, STRING, DATE, MATH, que son las que vamos a ver a continuación.
- **CLASES DEL BROWSER** (o navegador): las que tienen que ver con la navegación (document, window,...)
- **CLASES DEL HTML:** están asociadas con cualquier elemento de una página web(enlaces, tablas, formularios,...)

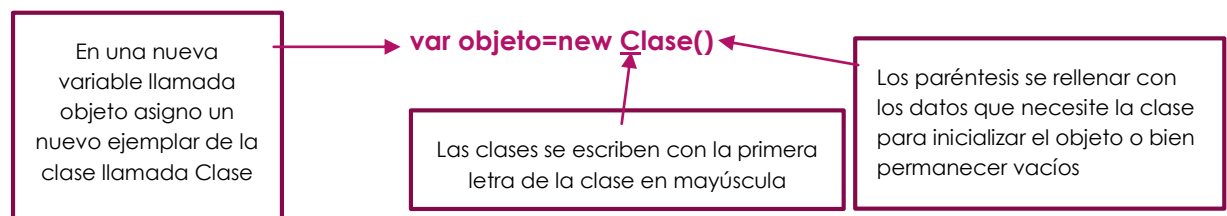
También podemos crear nuestras propias clases, con sus propiedades y sus métodos. Pero no es objeto de estudio en este curso. Un ejemplo, para que os hagáis una idea de su declaración y funcionamiento, lo podréis encontrar en este enlace:

<http://www.enrique7mc.com/2016/03/clases-en-javascript/>

## 9.2 ¿QUE ES UN OBJETO?

Un objeto es una instancia de una clase, es decir, se crean a partir de una clase. Es una entidad independiente que está definido por unas propiedades y un comportamiento, es decir, unas funciones o capacidades.

De esta forma, **instanciar un objeto** es la acción de crear un ejemplar de una clase para después poder trabajar con él y se realiza de la siguiente forma:



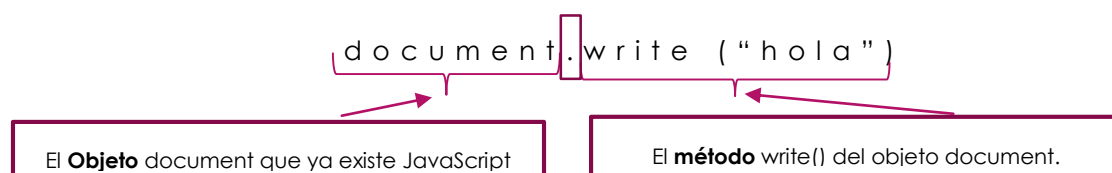
**Ejemplo:** `dia_sem= new Array ("lunes","martes",...,"domingo")`

(**dia\_sem** es el objeto de la clase **ARRAY**)

## 9.3 ACCEDER A LAS PROPIEDADES Y MÉTODOS

Vamos a ver la sintaxis para usar los objetos, es decir, cómo acceder a sus propiedades y ejecutar sus métodos.

Las propiedades definen el objeto y los métodos nos permiten operar él. Para acceder tanto a las propiedades como a los métodos utilizamos la **nomenclatura del punto**. Y esto nosotros ya lo hemos utilizado sin saber que estábamos haciendo uso de la POO, es decir, utilizando un objeto ya hecho en JavaScript llamado "document" que pertenece a la CLASE DEL BROWSER(navegador) junto con su método "write":



Otro ejemplo del que ya hemos hecho uso en Javascript es el método `alert()` del objeto `window`:

`window.alert()` o `window.prompt()`

Aunque el objeto `window` se puede omitir porque siempre que estamos programando lo estamos haciendo en una ventana (`window`) o para ella. Luego se puede omitir porque se sobreentiende que estamos trabajando con el objeto ventana

En este sentido, la nomenclatura del punto se utiliza de la siguiente forma:

	SINTAXIS	EJEMPLOS
PROPIEDADES	<b>Nombre_objeto . propiedad</b>	Para saber la longitud del Array utilizamos la propiedad <b>length</b> :  Dia_sem.length
	<b>Nombre_objeto . propiedad=Valor;</b>	<b>Ej1:</b> Modificar el ancho botón:  Botón.style.width=500px  <b>Ej2:</b> Tenemos un objeto llamado persona y queremos dar valores a cada una de las propiedades del objeto:  Persona.nombre="Antonio" Persona.apellido="lopez" Persona.edad=30 Persona.colorOjos="negro"
MÉTODOS	<b>Nombre_objeto.método();</b>  <b>Nombre_objeto.método(parametro1, parametro2,..)</b>	Añadir un elemento al final del Array con el método <b>push</b> :  Dia_sem.push("adios")

## 9.4 CLASES PREDEFINIDAS EN JAVASCRIPT

JavaScript dispone de varias clases predefinidas para acceder a muchas de las funciones normales de cualquier lenguaje, como son: Array (que ya hemos vistos en el apartado anterior), Date, Math, String. Cada uno de ellas quedará definida por unas propiedades y unos métodos, de forma que:

- ⇒ Si queremos trabajar con arrays, utilizaremos la clase ARRAY
- ⇒ Si queremos trabajar con cadenas de caracteres, utilizaremos la clase STRING
- ⇒ Si queremos trabajar con fechas, utilizaremos la clase DATE
- ⇒ Si queremos trabajar con funciones matemáticas, utilizaremos la clase MATH

### 9.4.1 CLASE STRING

Una cadena de caracteres (string) es uno o más caracteres encerrados entre comillas simples o dobles. Si creamos una variable con una frase estamos creando una cadena de caracteres o lo que es lo mismo, estamos creando un **objeto string**:

```
var apellido=new String() // var apellido=""
var apellido= new String("López") //var apellido="López"
```

Cada uno de los caracteres del String tiene un número de posición determinado:

i	0	1	2	3	4
apellido	l	o	p	e	z

NO confundid la clase STRING con la clase ARRAY. Un array es un conjunto de cuadraditos y cada uno puede almacenar distintos tipos de datos: números, caracteres, cadenas de caracteres. Sin embargo, un STRING es del tipo de datos cadena de caracteres y se almacena en una única variables:

#### STRING APELLIDOS:

0	1	2	3	4
l	o	p	e	z

#### ARRAY APELLIDOS

	0	1	2	3
	Lopez	Suárez	Martínez	Sánchez

PROPIEDADES	
<b>length</b>	Longitud de la cadena de caracteres
MÉTODOS	
<b>charAt(índice)</b>	Devuelve el <b>carácter</b> que hay en la posición indicada. Podemos utilizarlo para buscar un carácter en concreto dentro de una cadena de caracteres:
<pre> &lt;SCRIPT LANGUAGE='JavaScript'&gt; // Vamos a comprobar si el e_mail contiene la "@" var e_mail=prompt ("INTRODUCE TU EMAIL:","") for(var i=0; i&lt;=e_mail.length-1; i++) {     if (e_mail.charAt(i)=="@")         alert("CORRECTO")     else         alert("INCORRECTO") } &lt;/SCRIPT&gt; </pre>	<p>INTRODUCE TU EMAIL:</p> <p><a href="mailto:isabelsuarez@cotsalicane.com">isabelsuarez@cotsalicane.com</a></p> <p>CORRECTO</p>
<b>indexOf(carácter, desde)</b>	Devuelve la <b>posición</b> de la primera vez que aparece el carácter indicado. Si no lo encuentra devuelve -1. <i>Desde:</i> indica la posición desde donde empieza la búsqueda
<b>lastIndexOf(carácter, desde)</b>	Devuelve la <b>posición</b> de la última vez que aparece el carácter, pero la posición que devuelve es la correspondiente a empezar desde la izquierda
<pre> &lt;SCRIPT LANGUAGE='JavaScript'&gt; //Mostrar la posición de la @ en el e_mail var e_mail="isabelsuarez@cotsalicante.com" var posición=e_mail.lastIndexOf("@") document.write("POSICIÓN @:"+posición) &lt;/SCRIPT&gt; </pre>	<p>POSICIÓN @: 12</p>

<b>split(n)</b>	Devuelve un <b>array</b> con los elementos en que queda dividida la cadena según el separador n						
<pre>&lt;SCRIPT LANGUAGE='JavaScript'&gt; //Divide el texto de la cadena en dos tomando como separador la @ var e_mail="isabelsuarez@cotsalicante.com" var st=e_mail.<b>split</b>("@") document.write(st[0]+"&lt;br&gt;" +st[1]) &lt;/SCRIPT&gt;</pre>	<table><tr><th>posición</th><td>0</td><td>1</td></tr><tr><td>st</td><td>Isabelsuarez</td><td>Cotsalicante.com</td></tr></table> <p><b>SOLUCIÓN</b></p> <p>Isabelsuarez Cotsalicante.com</p>	posición	0	1	st	Isabelsuarez	Cotsalicante.com
posición	0	1					
st	Isabelsuarez	Cotsalicante.com					
<b>concat()</b>	Combina el texto de dos cadenas en una nueva						
<b>substring(inicio, fin)</b>	Devuelve un <b>fragmento de cadena</b> que empieza en <i>inicio</i> y termina en <i>fin</i> (sin incluir fin). Se utiliza para obtener el dominio de un email						
<pre>&lt;SCRIPT LANGUAGE='JavaScript'&gt; //Obtener el dominio var e_mail="isabelsuarez@cotsalicante.com" var inicio=e_mail.<b>lastIndexOf</b>("@") var fin=e_mail.length var dominio=e_mail.<b>substring</b>(inicio+1, fin) document.write(dominio) &lt;/SCRIPT&gt;</pre>							
<b>toLowerCase()</b>	Muestra cadena en minúsculas						
<b>toUpperCase()</b>	Muestra cadena en mayúsculas						
<b>toString()</b>	Este método lo tienen todos las clases y se usa para convertirlos en cadena						

### 9.4.2 CLASE DATE

Esta clase nos permite trabajar con fechas y horas. Para crear fechas se necesita instanciar un objeto de clase Date, y con él ya podemos realizar las operaciones deseadas.

Para crear un **objeto de la clase Date** lo haremos de dos maneras:

- Podemos crearlo con el día y hora actuales:

```
var fecha = new Date();
```

- Podemos crearlo con un día y hora concreto distintos al actual. Para ello les pasamos los parámetros de año, mes y día, que tienen que ser numéricos. Se debe tener en cuenta además, que los meses empiezan en el mes 0, de forma que Enero es el mes 0.

```
var fecha = new Date(año, mes, día);
```

```
var fecha = new Date(año, mes, día, hora, minuto, segundo);
```

La fecha se mide en milisegundos desde la media noche exacta del 01 de enero de 1970 en formato UTC. Un día contiene 86.400.000 milisegundos. El rango del objeto Date va desde -100,000,000 días hasta 100,000,000 días respecto del 01 de enero de 1970 UTC.

MÉTODOS	
<b>Date()</b>	Devuelve la fecha y hora de hoy
<b>getDate()</b>	Devuelve el día del mes actual
<b>getDay()</b>	Devuelve el día de la semana actual en forma de número empezando en el 0 (desde el 0 domingo y al 6 sábado)
<b>getMonth()</b>	Devuelve el mes actual. Los meses empiezan en 0 hasta el 11
<b>getFullYear()</b>	Devuelve el año actual pero de 4 dígitos
<b>getHours()</b>	Devuelve la hora actual
<b>getMinutes()</b>	Devuelve los minutos actuales. Entre 0 y 59
<b>getSeconds()</b>	Devuelve los segundos actuales. Entre 0 y 59
<b>getMilliseconds()</b>	Devuelve los milisegundos transcurridos según hora local
<div> <pre> &lt;SCRIPT type="javascript"&gt;   var fecha=new Date();   día=fecha.getDate();   mes=fecha.getMonth()+1;   año=fecha.getFullYear();   document.write(Date()+"&lt;br&gt;");   document.write("hoy es:"+día+"/"+mes+"/"+año); &lt;/SCRIPT&gt; </pre> </div> <div> <p>Resultado en el navegador:</p> <p><b>Tue Apr 25 17:37:25 2017</b>  <b>hoy es:25/4/2017</b></p> </div>	
<b>setDate(día)</b>	Actualiza el día del mes
<b>setMonth(mes, día)</b>	Establece el mes
<b>setFullYear(año, mes, día)</b>	Cambia el año de la fecha al año que recibe por parámetro (4 dígitos)
<b>setHours(hora, min, seg, miliseg)</b>	Actualiza la hora
<b>setMinutes(min,seg,miliseg)</b>	Cambia los minutos
<b>setSeconds(seg,miliseg)</b>	Cambia los segundos
<b>setMilliseconds(miliseg)</b>	Cambia los milisegundos

<b>setTime()</b>	Establece una fecha y hora añadiendo o restando un número determinado de milisegundos hasta/desde medianoche del 1 de enero de 1970
<b>toString()</b>	Convierte una fecha en una cadena de caracteres

Tened presente que estos últimos métodos, los métodos **"set..."** cambian los datos de tipo fecha y hora a nuevos datos y para volver a mostrarlos en pantalla escribimos solo el objeto como se indica en el ejemplo:

```
<SCRIPT language="javascript">
    var fecha = new Date(); //muestra la fecha de hoy
    document.write("La fecha de hoy es"+fecha+"<br>")
    var fecha30 = fecha.getDate(); //suma 30 días a la fecha de hoy
    fecha.setDate(fecha30+30);
    document.write("Dentro de 30 días será....."+fecha+"<br>")
</SCRIPT>
```

La fecha de hoy es.....Fri Apr 3 18:46:17 UTC+0200 2020  
 Dentro de 30 días será.....Sun May 3 18:46:17 UTC+0200 2020