



JavaScript



**ANEXO 1.-**

Fundamentos de Javascript



**DISTRIBUIDO POR:**

**CENTRO DE INFORMÁTICA PROFESIONAL S.L.**

C/ URGELL, 100  
08011 BARCELONA  
TFNO: 93 426 50 87

C/ RAFAELA YBARRA, 10  
48014 BILBAO  
TFNO: 94 448 31 33

[www.cipsa.net](http://www.cipsa.net)

**RESERVADOS TODOS LOS DERECHOS. QUEDA PROHIBIDO TODO TIPO DE  
REPRODUCCIÓN TOTAL O PARCIAL DE ESTE MANUAL, SIN PREVIO  
CONSENTIMIENTO POR EL ESCRITOR DEL EDITOR**

## *El lenguaje Javascript*

En un principio las páginas Web disponían de un contenido estático, tan sólo permitían que el usuario utilizase enlaces para moverse entre las distintas partes de su contenido. Con el paso del tiempo se ha visto la necesidad de desarrollar tecnologías que permitan dotar a las páginas de internet de interactividad, es decir, dar la posibilidad al usuario de interactuar con la información que se presenta en los navegadores.

HTML es un lenguaje declarativo que define la maquetación y el contenido de la página web. Este lenguaje permite mostrar diferentes elementos como párrafos, imágenes, hipervínculos..., anidados en otros elementos que hacen de contenedores como pueden ser las capas o las tablas.

Las CSS son declaraciones que definen el estilo visual de la página web estableciendo las características visuales de los diferentes elementos HTML que la componen. Las CSS han evolucionado para permitir que los elementos se adapten a las características del navegador (*diseño responsive*). Las CSS pueden definirse dentro de la propia página web o mediante ficheros externos (hojas de estilo) con extensión .css.

Javascript es un lenguaje de programación procedural que define el modo en que el usuario puede interactuar con la página web.

El concepto de interacción consiste en definir comportamientos para que se ejecuten antes ciertos eventos.

- Los *eventos* son sucesos y acciones del usuario sobre los elementos de la página que pueden emplearse para la ejecución de los comportamientos. Por ejemplo: la carga de la propia página, la pulsación de un botón..., etc.
- Los *comportamientos* son modificaciones en la página añadiendo, eliminando o alterando las características de los elementos HTML, que tienen lugar como respuesta a los eventos.

**Ejemplo:** Supóngase una página provista de un formulario en el que al pulsar el botón de envío (*evento*), se comprueba que los valores seleccionados e inscritos en el control del formulario sean correctos (comportamiento). Si todos son correctos se realiza el envío del formulario, y en caso contrario se muestra un mensaje de error destacando el campo incorrecto.

El código **JavaScript** puede ser incorporado directamente en la página web o ser inscrito en ficheros externos con extensión .js.

## Añadir Javascript a una página.

El código Javascript puede integrarse dentro del código de una página web o mediante un fichero aparte con extensión .js.

### Javascript integrado en la propia página.

El código Javascript se integra dentro del código de una página Web empleando la etiqueta de HTML: **<script>**. Las secciones de código de Javascript también se denominan “scripts”.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>new_file</title>
    <meta name="description" content="">
    <meta name="author" content="CIPSA">
    <meta name="viewport"
      content="width=device-width; initial-scale=1.0">
    <script>
      var i;
      for( i = 0; i <= 10; i++) {
        document.write("HOLA<br />");
      }
    </script>
  </head>
  <body>
  </body>
</html>
```

(\*) En HTML5 ya no es necesario incluir el atributo *type="text/javascript"* de la etiqueta **<script>** propio de las versiones anteriores de HTML.

### Javascript referenciado en un archivo .js

El código *Javascript* se incluye en un fichero separado con extensión **.js**, cuya ruta relativa debe incluirse en el atributo **src** de la etiqueta **<script>**:

#### Archivo: codigo.js

```
var mensaje = "HOLA";
var i;
for( i = 0; i <= 10; i++) {
  document.write( mensaje + "<br />");
}
```

Archivo: inicio.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>new_file</title>
    <meta name="description" content="">
    <meta name="author" content="CIPSA">
    <meta name="viewport"
      content="width=device-width; initial-scale=1.0">
    <script src="codigo.js"></script>
  </head>
  <body>
  </body>
</html>
```

Resultado visualizado:

HOLA  
HOLA  
HOLA  
HOLA  
HOLA  
HOLA  
HOLA  
HOLA  
HOLA  
HOLA  
HOLA

Si examinamos el código HTML de la página web veremos los mensajes "HOLA" que aparecen en pantalla NO aparecen en el código HTML. Esto es debido a que los mensajes han sido generados y agregados a la copia de la página presente en la memoria del navegador.

Por tanto, mediante Javascript podemos alterar una página web agregando y modificando sus elementos HTML; pero NUNCA modificar su código HTML original.

## Depuración de Javascript

La depuración de un código consiste en comprobar el modo en que se ejecuta para detectar el origen de errores y ver como resolverlos.

Los errores pueden ser de dos tipos:

- **Errores sintácticos** → Son errores que impiden que el código se ejecute. Estos errores se producen cuando se cometen errores al escribir el código.
- **Errores lógicos** → Son errores que hacen que el código no realice las operaciones deseadas a pesar de ejecutarse. Estos errores son producidos por fallos en el diseño el código que no al escribirlo.

La depuración debe realizarse en el propio navegador. Algunos navegadores como Chrome y IE incluyen herramientas para depurar el código Javascript, mientras que en otros como Firefox es necesario descargarlas en forma de plug-in (*FireBug*).


### Deteccion de errores sintácticos.

**Ejemplo:** La siguiente página consta de un código Javascript con un error deliberado en el que se define una función *saludar()* pero se invoca a la función *saludo()*. Este error al invocar a la función es un ejemplo de error sintáctico:

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
<title></title>
<meta charset="utf-8" />
<script>
    function saludar() {
        alert("HOLA");
    }
    saludo();
</script>
</head>
<body>
</body>
</html>
```

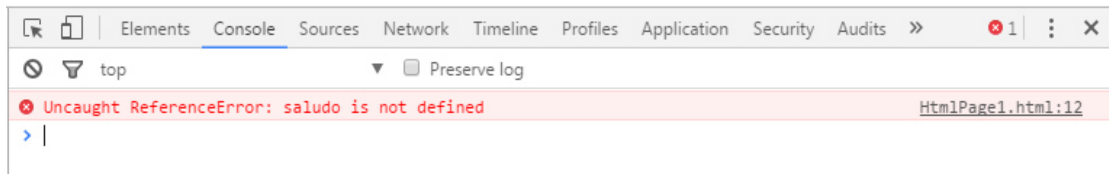
Si cargamos la página no veremos ningún mensaje ni tampoco mensaje de error alguno. En el caso de errores sintácticos el código simplemente no se ejecuta.

### Vista de errores sintácticos en Google Chrome.

El navegador Chrome integra una herramienta de desarrollo que se muestra seleccionando la opción **“Mas Herramientas → Herramientas para desarrolladores”** que se muestra al hacer clic en el icono  situado en el margen derecho de la barra de direcciones.

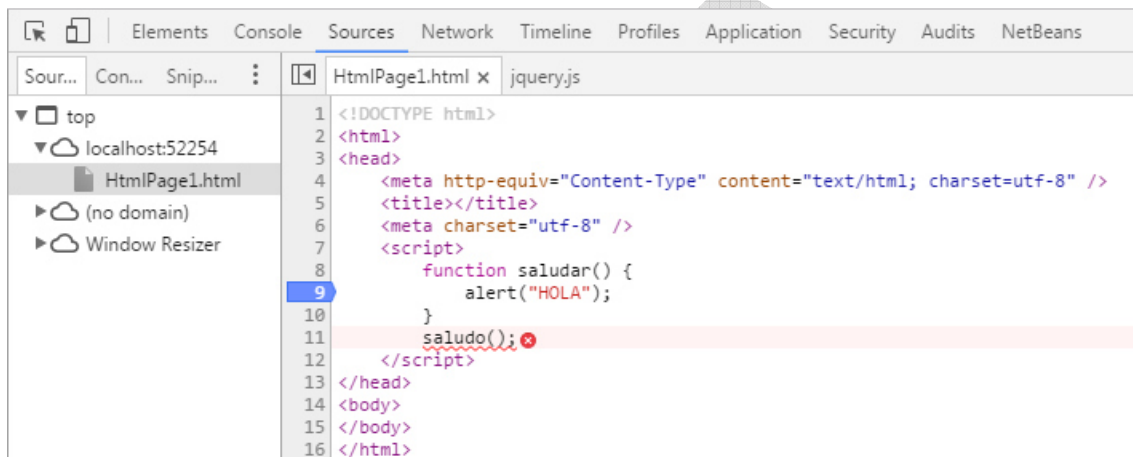
## CURSO DE PROGRAMACION JAVASCRIPT

Se abre entonces un cuadro provisto de varias pestañas. Los errores de ejecución de Javascript pueden consultarse seleccionando la pestaña “Console”:



*vista de consola indicando mensaje de error en Javascript*

Los errores sintácticos se muestran indicando un enlace en la derecha que al ser pulsado abre la pestaña *Sources* mostrando el código fuente con la línea de código origen del error destacada.

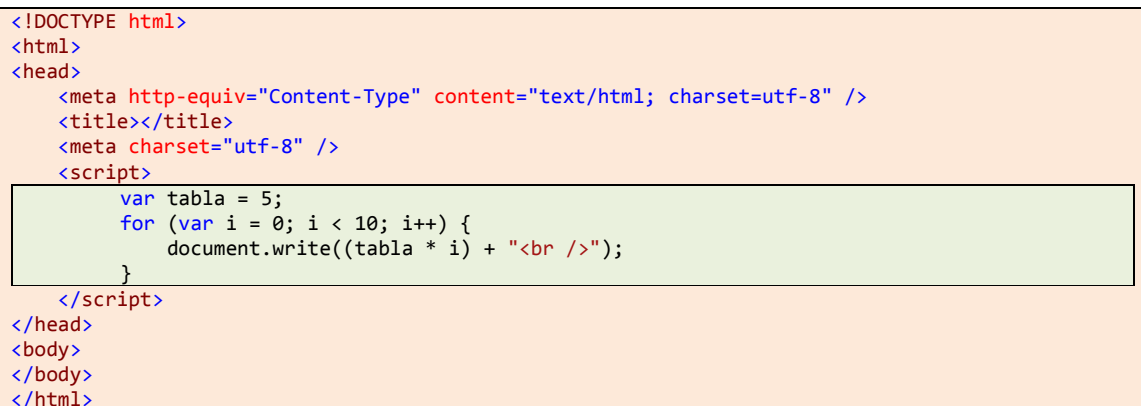


*Vista de código fuente indicando línea con error sintáctico*

### Deteccion de errores sintácticos.

Los errores lógicos no impiden que el código Javascript se ejecute, pero ocasionan que no funcione como se esperaba. En este caso no veremos ningún error en la consola, por lo que es necesario emplear otras técnicas para encontrar el origen del error.

**Ejemplo:** La siguiente página muestra la tabla de multiplicar del 5, del 0 al 9 cuando deseamos que lo haga del 0 al 10. Este es el caso de un error lógico en el que el código se ejecuta pero no hace lo que se desea:



## CURSO DE PROGRAMACION JAVASCRIPT

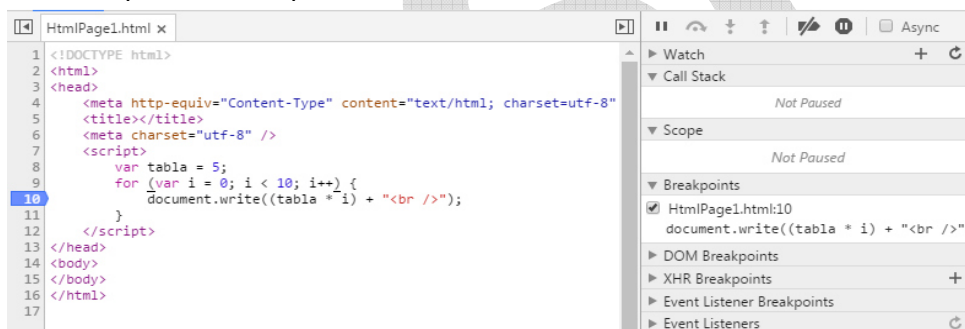
Para ayudar en la detección del origen de este tipo de errores se emplean dos técnicas:

- Trazado del código → Esto consiste en situar un punto de ruptura en una línea para que la ejecución se pause al alcanzarla. Una vez en pausa puede continuarse la ejecución del código línea a línea para ver el funcionamiento.
- Examen de variables → Esto consiste en comprobar el valor de las variables del código según se va trazando el código.

### Inserción de un punto de ruptura

Los puntos de ruptura se sitúan en aquellas líneas en las que queremos que se detenga la ejecución para comprobar el valor de las variables y comenzar a trazar la ejecución.

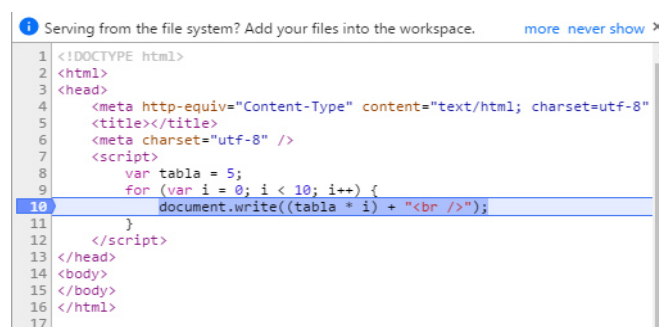
Para situar un punto de ruptura sólo es necesario hacer clic sobre su margen izquierdo. La línea se destaca entonces con un indicador azulado. Para eliminar un punto de ruptura basta con hacer clic en el marca al margen derecho. En un código pueden ponerse tantos puntos de ruptura como se desee:



*Código con un punto de ruptura situado en la línea 10*

Todos los puntos de ruptura presentes en el código se muestran bajo la pestaña "Breakpoints" del margen derecho. La casilla de verificación permite habilitar o deshabilitar un punto de ruptura para evitar que pause la ejecución sin eliminarlo.

Una vez situado los puntos de ruptura al recargar la página pulsando la tecla enter sobre la barra de direcciones, la ejecución se detiene al alcanzar el primer punto de ruptura. La línea queda entonces resaltada en azul:



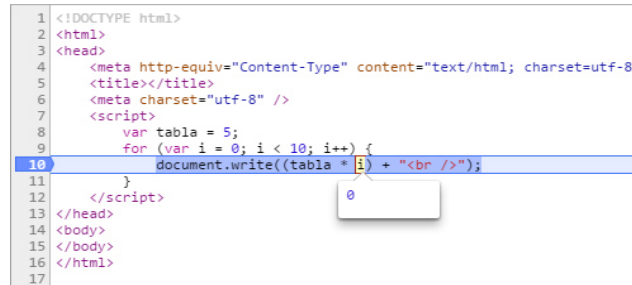
*Código con ejecución detenida en punto de ruptura*



### Examinar variables

El examen de variables consiste en comprobar el valor de las variables presentes en el código mientras la ejecución está pausada. Esto puede hacerse de dos modos:

- Situando el puntero del ratón directamente encima de la variable. Se muestra entonces un cuadro flotante indicando el valor:



*Visual del valor de la variable "i" situando el ratón encima*

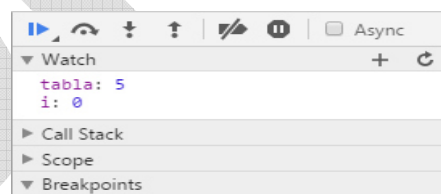
- Añadiendo las variables a la pestaña Watch haciendo clic en el botón "+" situado en el margen derecho:







*Pestaña de inspección con las variables 'tabla' y 'i'.*

### Seguimiento del código

Una vez que la ejecución está detenida en un punto de ruptura podemos seguir el código ejecutándolo línea a línea mediante los botones de la barra de herramientas de la parte superior del margen derecho:



- **Step Over** (  ) → Ejecuta la siguiente línea de código.
- **Step Into** (  ) → Ejecuta la siguiente línea de código. Si la línea es una llamada a una función, se salta a trazar a la primera línea del código de la función.
- **Step Out** (  ) → Salta a la línea de código siguiente a la llamada a la función actual.
- **Resume** (  ) → Reanuda la ejecución hasta el siguiente punto de ruptura.

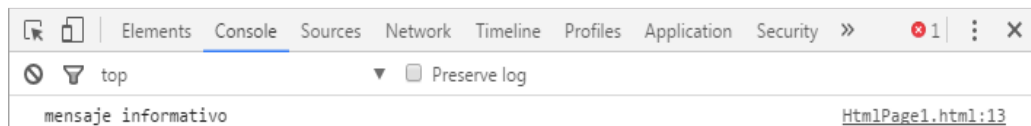
Al ir ejecutando cada línea del código veremos como la línea actual de ejecución va cambiando destacada en azul. Al mismo tiempo puede comprobarse como van progresando el valor de las variables de la pestaña *Watches*.

### Lanzamiento de mensajes a consola

Como mecanismo adicional para verificar la ejecución de Javascript también pueden enviarse mensajes a la consola del navegador desde el propio código mediante los métodos del objeto de javascript *console*:

- `console.log( mensaje )` → Lanza a la consola el mensaje indicado.

```
console.log("mensaje informativo");
```



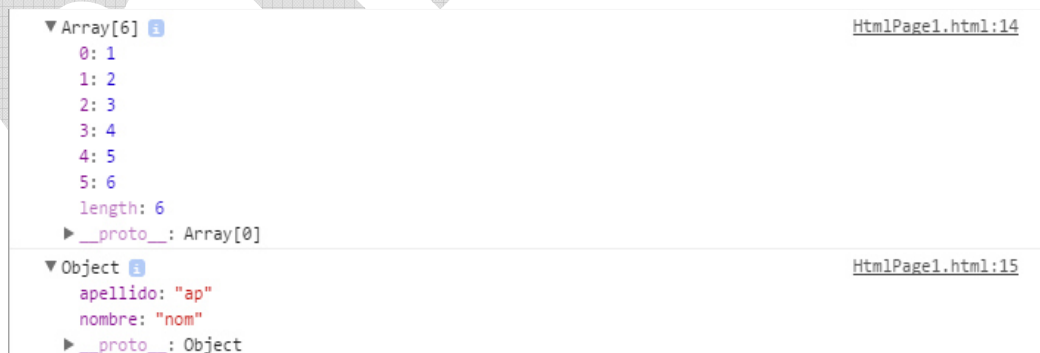
- `console.error( mensaje )` → Lanza a la consola el mensaje de error indicado incluyendo además la pila de llamadas que ha recorrido la ejecución para llegar a esa línea de código.

```
console.error("Error en sistema");
```



- `console.dir( objeto )` → Lanza a la consola una descripción del tipo y valor o valores contenidos en la variable, matriz u objeto indicado como argumento.

```
var datos = [1, 2, 3, 4, 5, 6];  
var obj = {  
  nombre: "nom",  
  apellido: "ap"  
}  
console.dir(obj);
```



## Sintaxis Básica

### Variables

Las variables son elementos del lenguaje que permiten almacenar distintos valores en cada momento. Se puede almacenar un *valor* en una variable y consultar este valor posteriormente, también podemos modificar su contenido siempre que queramos.

Las variables deben ser siempre declaradas antes de ser utilizadas mediante la palabra clave **var** y asignando un valor inicial:

```
var foo = 'hola mundo';
```

Los identificadores de las variables deben cumplir ciertas reglas para ser válidos:

- Deben empezar por una letra o un guión bajo, y el resto de los caracteres siguientes pueden ser letras o números.
- Las variables no pueden llamarse igual que una palabra reservada del lenguaje tales como por ejemplo: **var**, **for**, **while**, **if**....
- No se admiten espacios en blanco en el nombre de una variable.
- No se permiten letras acentuadas ni otras que no pertenezcan al alfabeto internacional como es el caso de ñ, ç.. etc.
- El nombre de las variables es sensible a las mayúsculas. Esto quiere decir que Javascript considera distintas a variables cuyos nombres se diferencian en el uso de las mayúsculas y minúsculas; p.ej: “Edad” y “edad”.

```
// Variables correctas
var _Una_Variable, P123robando, _123, mi_carrooo;

// Variables no válidas
var Mi Variable, 123Probando, $Variable, for, while;
```

Una vez declarada una variable puede asignársele un valor tantas veces como se quiera. Cada valor asignado elimina el anteriormente existente.

```
foo = 'hola mundo 2';
```

### Tipos de valores

Las variables en Javascript pueden ser de diferentes tipos según el valor que se les asigne:

- **Numéricas:** Son aquellas a las que se asignan números enteros o decimales.
- **Alfanuméricas:** Son aquellas que reciben cadenas de texto entre comillas.
- **Lógicas:** Son un tipo especial de variables que contienen un valor lógico cierto o falso. Para darles valor se emplean las palabras clave **true** y **false**, o 0 y 1.

El tipo de dato de una variable depende del valor que se le asigne.

Tipo de variable	Asignaciones
Númerica	<pre>numero1 = 3; numero2 = 3.7; numero3 = 0.6; numero4 = 5+7;</pre>
String	<pre>cadena1 = "Pepe"; cadena2 = "Bienvenido a el site Web";</pre>
Boolean	<pre>entrada = true; salida = false; auxentrada = 1; auxsalida = 0</pre>

## Operadores

Los operadores permiten manipular el valor de las variables según sea su tipo:

### Operadores aritméticos:

Los operadores aritméticos permiten ejecutar diferentes tipos de operaciones entre variables y valores de tipo numérico. Los operadores aritméticos se emplean entre variables numéricas y el resultado es así mismo un valor numérico:

Operador	Nombre	Ejemplo	Descripción
+	Suma	5 + 6	Suma dos números
-	Substracción	7 - 9	Resta dos números
*	Multiplicación	6 * 3	Multiplica dos números
/	División	4 / 8	Divide dos números
%	Módulo: el resto después de la división	7 % 2	Devuelve el resto de dividir ambos números, en este ejemplo el resultado es 1
++	Incremento.	a++	Suma 1 al contenido de una variable.
--	Decremento.	a--	Resta 1 al contenido de una variable.

Ejemplos de expresiones aritméticas:

```
var a = 10;
var b = 20;
alert( a + b );      // Muestra 30
alert( a - b );      // Muestra -10
alert( a * b );      // Muestra 200
alert( a / b );      // Muestra 0.5
a++;                // Incrementa el valor de la variable en 1
alert( a );          // Muestra 11
```

Los paréntesis representan prioridad en la evaluación de expresiones:

```
var x = 2 + 3 * 6;    // x vale 20
var y = (2 + 3) * 6;  // y vale 30
```

### Operador de concatenación

El operador de concatenación concatena dos cadenas de texto:

```

var nombre = "pedro";
var apellido = "ramirez";
var nombreCompleto = nombre + " " + apellido;
alert( nombreCompleto ); // Muestra "pedro ramirez" en pantalla

```

Hay que tener cuidado con las operaciones entre cadenas y números:

```

var a = 1;
var b = "1";
// numero + cadena --> CONCATENACION
alert(a + b);           // Muestra por pantalla 11

// numero + numero --> SUMA
alert( a + Number(b))   // Muestra por pantalla 2

```

(\*) La función **Number()** retorna el valor numérico equivalente a un valor cadena.

### Operadores de comparación

Los operadores de comparación permiten comparar variables y valores devolviendo valor *cierto* o *falso*. Estos operadores se utilizan para condiciones que permiten tomar de decisiones en Javascript:

Operador	Descripción
<b>==</b>	" <b>Igual a</b> " devuelve true si los valores son iguales.
<b>!=</b>	" <b>distinto de</b> " devuelve true si los valores son diferentes
<b>&gt;</b>	" <b>mayor que</b> " devuelve true si el valor de la izquierda es mayor que el de la derecha.
<b>&gt;=</b>	" <b>mayor o igual que</b> " devuelve true si el valor de la izquierda es mayor o igual que el valor de la derecha.
<b>&lt;</b>	" <b>menor que</b> " devuelve true si el valor de la izquierda es menor que el de la derecha.
<b>&lt;=</b>	" <b>menor o igual que</b> " devuelve true si el valor de la izquierda es menor o igual que el de la derecha.

Ejemplo de expresiones condicionales:

```

var a = 10;
alert( a == 10 );      // Muestra "True"
alert( a > 10 );       // Muestra "False"

```

**Operadores lógicos**

Los operadores lógicos se emplean para combinar condiciones. Existen tres tipos de operadores lógicos:

Operador	Descripción
&&	Conjunción "Y" → Cierta si todas las condiciones son ciertas.
	Disyunción "O" → Falso si todas las condiciones son falsas.
!	Negación → Hacer falso lo cierto y cierto lo falso.

*Ejemplo:* La siguiente expresión devuelve cierto si la variable 'num' tiene un valor comprendido entre 20 y 30 ambos incluidos:

```
num >= 20 && num <= 30
```

*Ejemplo:* La siguiente expresión se evalúa como cierta si la variable 'nombre' tiene como valor "pedro" o "luis":

```
nombre == "pedro" || nombre == "luis"
```

**Operadores de Asignación**

Los operadores de asignación se emplean para modificar el valor de una variable mediante otro valor o variable:

Operador	Descripción
=	Asigna el valor del operando de la derecha a la variable de la izquierda.
+=, -=, *=, /=	Añade el valor del operando de la derecha a la variable de la izquierda.

*Ejemplo:*

```
valor += numero; // Equivale a: valor = valor + numero.
valor -= numero; // Equivale a: valor = valor - numero.
```

**Ejercicio propuesto:**

Haz que la página muestre una línea indicando la edad que tendrá el usuario el año próximo.

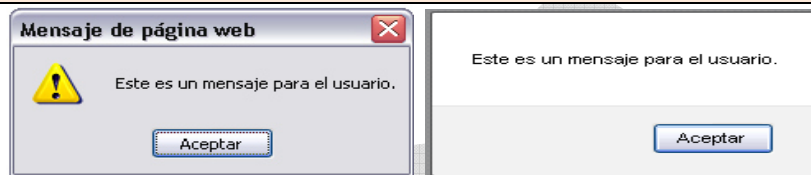
## Cuadros de diálogo básicos

Javascript permite lanzar cuadros de diálogo desde una página con tres finalidades según su tipo: notificarle un suceso, solicitarle una confirmación, o requerir un valor:

### Función alert().

Esta función muestra un cuadro de diálogo informativo con el mensaje indicado como argumento:

```
alert("Este es un mensaje para el usuario");
var mensaje = "Este es un mensaje para el usuario";
alert( mensaje );
```

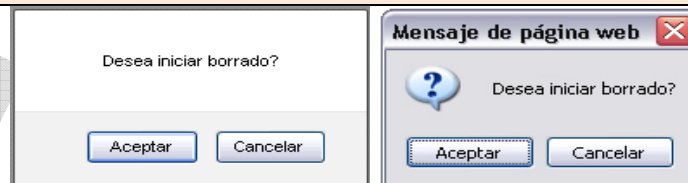


*Cuadro de diálogo de Alert() en Internet Explorer ( Izquierda ) y en Firefox ( Derecha )*

### Función confirm().

Esta función muestra un cuadro de diálogo con el mensaje indicado solicitando una respuesta afirmativa o negativa mediante dos botones. Si se pulsa el botón “Aceptar” la función devuelve un valor “true”, en caso contrario devuelve “false”:

```
var respuesta = confirm("Desea iniciar borrado?");
alert( respuesta );
```

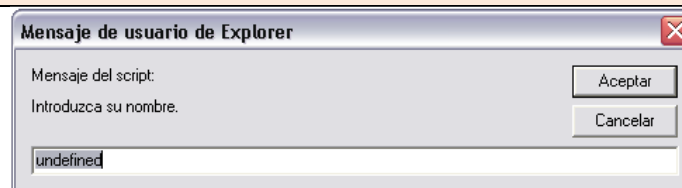


*Cuadro de diálogo de Confirm() en Internet Explorer ( Izquierda ) y en Firefox ( Derecha )*

### Función prompt().

Esta función muestra un cuadro de diálogo con el mensaje indicado solicitando al usuario un valor mediante una caja de texto. La función devuelve el valor introducido:

```
var nombre = prompt("Introduzca su nombre.");
alert( nombre );
```



*Cuadro de diálogo de petición de información en Internet Explorer*

La función **prompt()** devuelve siempre una cadena de caracteres incluso aunque el usuario introduzca un valor numérico. Si se va a realizar alguna operación numérica con el valor introducido es preciso convertirlo primero a un valor numérico.

**Ejemplo:** Supóngase el siguiente código de Javascript:

```
var edad = prompt("Dime cuanto años tienes", "");  
edad = edad + 1;  
document.write("El año que viene tendrás: " + edad + " años");
```

Si el usuario introduce el valor "20", la página mostrará el siguiente mensaje:

*El año que viene tendrás: 201 años*

Para resolverlo debe convertirse el valor de la variable *edad* en número mediante la función **Number()** antes de hacer la suma:

```
var edad = prompt("Dime cuanto años tienes", "");  
edad = Number( edad );  
edad = edad + 1;  
document.write("El año que viene tendrás: " + edad + " años");
```

El resultado mostrado ahora es:

*El año que viene tendrás: 21 años*

### Ejercicio Propuesto

Crea un código Javascript con las sentencias necesarias para realizar las siguientes operaciones:

- ✓ Declara una variable *precio\_manzana* con el valor 2
- ✓ Se solicita al usuario los kilos de manzanas comprados almacenando el valor en la variable *kilos\_manzanas*.
- ✓ Declara una variable *precio\_naranja* con el valor 2.5
- ✓ Se solicita al usuario los kilos de naranjas comprados almacenando el valor en la variable *kilos\_naranjas*
- ✓ Declara una variable *precio\_platano* con el valor 1.75
- ✓ Se solicita al usuario los kilos de plátanos comprados almacenando el valor en la variable *kilos\_platanos*.
- ✓ Declara una variable *compra*.
- ✓ Incrementa la variable *compra* con el producto de las variables *precio\_manzana* y *kilos\_manzanas*.
- ✓ Incrementa la variable *compra* con el producto de las variables *precio\_naranja* y *kilos\_naranjas*.
- ✓ Incrementa la variable *compra* con el producto de las variables *precio\_platano* y *kilos\_platanos*.
- ✓ Declara una variable *oferta* cuyo valor sea 0.08
- ✓ Declara una variable *descuento* cuyo valor sea el producto de las variables *compra* y *oferta*.
- ✓ Modifica el valor de la variable *compra* restándole el valor de la variable *descuento*.
- ✓ Muestra el valor de la variable *compra* con el mensaje "El precio de su compra es: <valor\_compra> Euros".



## Estructuras de Control

El código Javascript se ejecute por defecto secuencialmente. Las estructuras de control alteran el orden de ejecución por defecto permitiendo que partes del código se ejecuten bajo ciertas condiciones o varias veces.

### Estructura condicional ( *if – else* )

Las estructuras condicionales permiten definir secciones de código que se ejecutan si una determinada condición es cierta o falsa:

```
if (<condición>) {
    codigo que se ejecuta si la condición es cierta.
} else {
    código que se ejecuta si la condición es falsa.
}
```

La **<condición>** puede ser una expresión condicional:

- ✓ `if ( valor > 18 )`
- ✓ `if ( valor >= 18 && valor <= 32 )`
- ✓ `if ( nombre == "pedro" || nombre == "luis" )`

También puede utilizarse como condición una variable con un valor *cierto* o *falso*.

```
if ( ok ) {
    // Código que se ejecuta si ok = true
} else {
    // Código que se ejecuta si ok = false
}
```

**Ejemplo:** El siguiente código solicita al usuario que introduzca su edad y le muestra un mensaje informativo en función de si es o no mayor de edad.

```
var edad;
edad = prompt("Qué edad tiene?");
edad = parseInt( edad );
if ( edad > 18 ) {
    // RAMA CIERTO -> Sólo se ejecuta si la condición es cierta
    alert("Es usted mayor de edad.");
} else {
    // RAMA FALSO -> Sólo se ejecuta si la condición es falsa
    alert("Es usted menor de edad.");
}
```

(\*) El tabulado del código contenido en cada rama de la condicional facilita su legibilidad sin afectar a la ejecución.

El bloque **else** puede omitirse si no es necesario ninguna sección de código para el caso en que la condición no sea cierta.

**Ejercicios Propuestos:**

1. Crea una página que solicite al usuario el nombre del día de la semana. Si el valor indicado es "sábado" o "domingo", la página debe mostrar un cuadro de diálogo con el mensaje "Es Festivo.". En caso contrario debe indicarse el mensaje "Es laboral.".
2. Crea una página que solicite el nombre de dos personas y sus respectivas edades y después muestre el mensaje:
  - ✓ "<nombre\_persona\_A> es mayor que <nombre\_persona\_B>" → Si una persona es mayor que otra.
  - ✓ "Ambas personas tienen la misma edad" → Si la edad de ambas personas es la misma.

**Operador condicional ternario ( ? )**

Este operador permite asignar a una variable un valor u otro en función de una condición:

**( <condición> ) ? <valor si cierto> : <valor si falso >**

**Ejemplo:** El siguiente código asigna a la variable mensaje un valor "Es mayor" o "Es menor" en función de si el valor de la variable edad es superior a 18.

```
var edad = prompt("Cual es su edad", "");
var mensaje;
edad = parseInt( edad);

if ( edad >= 18 ) {
    mensaje = "Es mayor";
} else {
    mensaje = "Es menor";
}

document.write(mensaje);
```

Puede simplificarse empleando el operador condicional ternario del siguiente modo:

```
var edad = prompt("Cual es su edad", "");
edad = parseInt(edad);

var mensaje = ( edad > 18 ) ? "Es mayor" : "Es menor";

document.write(mensaje);
```

### Ejercicios Propuestos

1. Crea una página que solicite al usuario su nombre. En caso de que el nombre indicado sea "Roger" se mostrará un mensaje dándole la bienvenida. En caso contrario se mostrará el mensaje "Página no disponible".
2. Crea una página que solicite al usuario un nombre y contraseña. Si el nombre es "Roger" y la contraseña es "CIPSA" se mostrará un mensaje de bienvenida.
3. Si el nombre de usuario es incorrecto se mostrará el mensaje de error "Usuario no reconocido.". Si el nombre de usuario es correcto pero no la contraseña se mostrará el mensaje "Contraseña incorrecta".
4. Crea una página que solicite al usuario el nombre del mes y día actual. Si el usuario introduce el día 25 de "diciembre" deberá mostrarse el mensaje "Es Navidad".
5. Crea una página que solicite al usuario un valor numérico. Si el valor introducido es múltiplo de 2 deberá mostrarse el mensaje "El número es par". En caso contrario deberá mostrarse el mensaje "El número es impar".
6. Crea una página que solicite al usuario el número de preguntas existentes en un examen de tipo Test, y cuántas ha acertado. En función de los valores indicados la página deberá calcular la nota obtenida y mostrar la calificación que corresponda en función de los siguientes baremos:
  - ✓ Si la nota está entre 0 y 2 → *Muy deficiente*
  - ✓ Si la nota está entre 2 y 5 → *Insuficiente*
  - ✓ Si la nota está entre 5 y 6 → *Suficiente*
  - ✓ Si la nota está entre 6 y 7 → *Bien*
  - ✓ Si la nota está entre 7 y 9 → *Notable*
  - ✓ Si la nota está entre 9 y 10 → *Sobresaliente*.
7. Crea una página que solicite alternativamente el nombre y edad de tres personas y muestre a continuación el nombre de la mayor y la menor de ellas.
8. Crea una página que solicite al usuario su nombre. Si el usuario introduce su nombre y pulsa el botón "Aceptar" deberá mostrarse un mensaje de bienvenida "Encantado de conocerte, XXXX". Si el usuario no introduce ningún nombre o cierra el cuadro de diálogo sin responder deberá mostrarse el mensaje "Usuario anónimo".

## Estructura condicional de múltiple rama ( *switch* )

La estructura ***switch*** define múltiples bloques de código (ramas) asociadas a un valor de tipo *entero* o *cadena*. Sólo se ejecuta la rama cuyo valor coincide con el de la variable de control. En caso contrario no se ejecuta ninguna.

```
switch ( variable )
{
    case valor_1:
        <instrucciones>
        break;
    case valor_2:
        <instrucciones>
        break;
    case valor_3:
        <instrucciones>
        break;
    default:
        <instrucciones>
}
```

La rama ***default*** es opcional. En caso de indicarse se ejecuta únicamente si el valor de la variable de control no coincide con el de ninguna rama.

**Ejemplo:** El siguiente código solicita al usuario el número del día de la semana y muestra su nombre correspondiente:

```
var dia;
dia = prompt("Indica numero del mes", "");
dia = parseInt( dia );
switch ( dia ) {
    case 1: {
        document.write("Lunes");
    }; break;
    case 2: {
        document.write("Martes");
    }; break;
    case 3: {
        document.write("Miercoles");
    }; break;
    case 4: {
        document.write("Jueves");
    }; break;
    case 5: {
        document.write("Viernes");
    }; break;
    case 6: {
        document.write("Sabado");
    }; break;
    case 7: {
        document.write("Domingo");
    }; break;
    default: {
        document.write("Valor no válido");
    }
}
```

**Ejemplo:** El siguiente código solicita al usuario un resultado de quiniela y muestra el valor correspondiente:

```
var opcion;
opcion = prompt("Indica el resultado.", "");
switch (opcion) {
  case "1":
    alert("uno");
    break;
  case "2":
    alert("dos");
    break;
  case "X":
    alert("equis");
    break;
  default:
    alert("desconocido.");
    break;
}
```

## Condicionales anidadas

Los bloques de una estructura condicional pueden contener estructuras condicionales anidadas:

```
If ( condición_1 ) {
    Instrucciones si cumple condicion_1
} else if ( condición_2 ) {
    Instrucciones si cumple condicion_2
} else if ( condición_3 ) {
    Instrucciones si cumple condicion_3
} else {
    Instrucciones si no cumple ninguna de las condiciones anteriores
}
```

**Ejemplo:** El siguiente código solicita al usuario la temperatura del agua e indica a continuación su estado físico correspondiente:

```
var temperatura;
temperatura = prompt("¿Qué temperatura tiene el agua?", "");
temperatura = parseFloat( temperatura);
if( temperatura <= 0.0) {
    // Si la temperatura es inferior o igual a 0.
    document.write("HIELO");
} else if ( temperatura > 0.0 && temperatura < 100.0) {
    // Si la temperatura es mayor que 0 a inferior a 100
    document.write("LIQUIDO");
} else {
    // Si la temperatura es igual o mayor que 100
    document.write("VAPOR");
}
```

### ***Ejercicios Propuestos***

1. Crea una página que solicite al usuario dos valores numéricos y una operación “suma”, “resta”, “producto” o “división”. La página mostrará a continuación el resultado correspondiente en función de la operación solicitada.

Si el usuario introduce una palabra diferente a las indicadas deberá mostrarse el mensaje “Operación no válida”.

2. Modifica el ejercicio anterior de modo que si el usuario introduce un 0 como segundo valor y elige como operación una división se muestre el mensaje “División por cero”.

## Estructura repetitiva ( *for* )

La estructura repetitiva ( o *bucle* ) *for* permite la ejecución de una sección de código un determinado número de veces.

```
for( <inicialización> ; <condición> ; <modificación> ) {
    Instrucciones
}
```

Esta estructura emplea una variable de control para gestionar el número de veces que debe ejecutarse el código en función de las siguientes expresiones:

- <inicialización> → Se ejecuta únicamente la primera vez para asignar un valor inicial a la variable de control
- <condición> → Se evalúa antes de cada repetición. Sólo se ejecuta el código si la condición es cierta.
- <modificación> → Se ejecuta tras cada repetición alterando el valor de la variable de control.

**Ejemplo:** El siguiente código muestra en la página los números comprendidos entre el 1 y 10 ambos incluidos:

```
for( var veces = 1; veces <= 10; veces++) {
    // Muestra en la página el valor de la variable veces
    document.write( veces + "<br >");
}
```

En la inicialización ( **var** veces = 1 ) se asigna el valor inicial a la variable de control. El bucle se repite mientras la condición de prueba ( veces <= 10 ) es cierta. Tras cada ejecución del bucle se aplica la modificación que incrementa la variable de control en la unidad ( veces++ ).

### Ejercicio resuelto:

Crea una aplicación que solicite al usuario un valor numérico mayor que 0, y después muestre todos comprendidos entre el valor dado y el 0 en orden decreciente. Si el usuario introduce el valor 10 deberán mostrarse los valores: 9,8,7,6,5,4,3,2,1 y 0:

```
var valor;
// Se solicita el valor al usuario
valor = prompt("Introduce un valor mayor que 0");
valor = parseInt( valor );
if ( valor > 0 ) {
    // Si el valor es mayor que 0 se ejecuta el bucle
    for( var numero = valor; numero >= 0; numero-- ) {
        document.write(numero + "<br >");
    }
} else {
    // Si el valor es igual o menor que cero se muestra error.
    document.write("El valor indicado no es valido");
}
```

## Estructura repetitiva ( *while* )

Un bucle **while** ejecuta una sección de código mientras la condición es cierta. En cierto modo es semejante a una estructura condicional con la diferencia que el código se repite hasta que la condición deje de evaluarse como cierta:

```
while ( condición ) {
    sentencias
}
```

La condición se evalúa antes de ejecutar el bucle, por lo que si resulta falsa de primeras el bucle no se ejecuta ni una vez.

**Ejemplo:** El siguiente código solicita al usuario una clave. Si la clave es diferente de la contraseña se le muestra un mensaje de error y se le vuelve a solicitar. El proceso se repite hasta que indique la clave correcta.

```
var contraseña = "CIPSA";
var clave;

clave = prompt("Introduce la contraseña", "");

while ( clave != contraseña){
    alert("Error. Clave incorrecta");
    clave = prompt("Introduce la contraseña", "");
}
```

## Estructura repetitiva ( *do - while* )

Un bucle **while** ejecuta una sección de código mientras la condición es cierta. La diferencia con la estructura anterior es que la condición se evalúa tras la ejecución del bucle, por lo que si es falsa de principio el bucle se ejecuta al menos una vez.

```
do {
    sentencias
} while ( condición )
```

**Ejemplo:** El siguiente código solicita al usuario una clave. Si la clave no coincide con la contraseña se le vuelve a solicitar hasta que la indique correctamente.

```
var contraseña = "CIPSA";
var clave;
do
    clave = prompt("Introduce la contraseña", "");
while ( clave != contraseña)
```



### Las instrucciones *break* y *continue*

Un bucle normalmente se ejecuta mientras su condición se evalúa como cierta.

La sentencia ***break*** provoca el fin del bucle independientemente de su condición.

*Ejemplo:* Sea el siguiente bucle que debería dar 10 vueltas mostrando los valores del 0 al 9. Sin embargo, los valores mostrados son 0,1,2,3,4. Cuando x vale 5 se ejecuta la sentencia ***break*** cancelando el bucle.

```
for( var x = 0; x < 10; x++) {  
    if ( x == 5 ) break;  
    document.write( x + "<br >");  
}
```

La sentencia ***continue*** provoca el salto al principio del bucle saltándose el resto del código tras su ejecución:

*Ejemplo:* El siguiente bucle debería dar 10 vueltas y mostrar los valores del 0 al 9, pero se muestran los valores 0,1,2,3,4,6,7,8,9. Falta el valor 5 ya que se ejecuta la sentencia ***continue*** y fuerza al bucle a volver al principio ignorando las líneas siguientes.

```
for( var x = 0; x < 10; x++) {  
    if ( x == 5 ) continue;  
    document.write( x + " ");  
}
```

### Ejercicios Propuestos

1. Crea una aplicación que muestre los números pares comprendidos del 100 al 0 ambos incluidos empleando los tres tipos de bucles ***for***, ***while*** y ***do-while***.
2. Crea una aplicación que solicite al usuario un valor numérico. Si el usuario introduce un valor no válido como letras o texto; deberá mostrarse un cuadro de diálogo con el mensaje "ERROR" y volvérselo a solicitar el valor hasta que lo indique correctamente. Para hacer este ejercicio debe emplearse la función ***isNaN()*** de Javascript.
3. Crea una aplicación que solicite al usuario valores mostrando a continuación un cuadro de diálogo de confirmación mediante la función ***confirm()*** preguntándole si quiere continuar. Una vez que termina la página deberá mostrar al usuario la cantidad de valores introducidos y su media.
4. Crea una aplicación que solicite al usuario un valor entre 1 y 10. La página solicitará después uno a uno los valores de la tabla de multiplicar del valor indicado inicialmente. Si el usuario acierta cada valor se le mostrará un cuadro de diálogo con el mensaje "CORRECTO". En caso contrario se le mostrará un cuadro de diálogo indicando el valor correcto. Una vez preguntados todos los valores la página deberá mostrar el porcentaje de aciertos del usuario.

## Funciones y ámbitos de variables

Una función es una sección de código que realiza una determinada tarea necesaria en diferentes momentos. Pueden recibir una serie de argumentos para realizar su tarea y opcionalmente devolver un resultado.

### Declaración

Declaración tradicional.

```
function <identificador> ( <parametro1>, <parametro2>, .... ) {
    <sentencias>
    return <valor_retorno>      <← *opcional* >
}
```

Declaración de función nominada → La función se declara como el valor de una variable mediante una asignación.

```
var <identificador> = function ( <parametro1>, <parametro2>, .... ) {
    <sentencias>
    return <valor_retorno>      <← *opcional* >
}
```

Elementos integrantes de la declaración:

- ✓ **<identificador>** → Es el nombre de la función que se emplea para llamarla.
- ✓ **<parámetros>** → Son variables para los valores que la función debe recibir cuando es llamada para realizar su tarea.
- ✓ **<valor\_retorno>** → Es el valor que retorna como función resultado. Debe indicarse a la derecha de la sentencia **return**. Es opcional.

### Llamada

Toda función se llama por su identificador indicando un valor o variable (argumentos) para cada parámetro. Si la función no tiene parámetros no se indica nada entre los paréntesis:

```
<identificador>( <argumento1>, <argumento2>, ... )
```

*Ejemplo:* La siguiente función *ahora()* muestra un cuadro de diálogo indicando la fecha actual del sistema. La función no requiere ningún parámetro:

```
// Declaración de la función
var ahora = function() {
    alert(new Date());
}

// Llamada a la función
ahora();
```

*Ejemplo:* Sea la función *saludar()* que muestra un cuadro de diálogo mostrando un mensaje de saludo al usuario con el nombre y origen requeridos como parámetros:

```
// Declaración de la función
var saludar = function (nombre, lugar) {
    alert("Te saludo " + nombre + " procedente de " + lugar);
}
// llamada con valores.
saludar("Roger", "América");
// llamada con variables
var name = "Roger";
var origen = "América";
saludar(name, origen);
```

Si la función devuelve un valor se la llama de manera idéntica pero a la derecha de una asignación donde la variable a la izquierda recoge el valor devuelto por la función:

**<variable> = <identificador>( <argumento1>, <argumento2>, ... )**

*Ejemplo:* La función *media()* devuelve la media aritmética de los dados a la función.

```
// Funcion media
var media = function(a, b, c) {
    return (a + b + c) / 3;
}
var a1 = 10;
var a2 = 20;
var a3 = 30;
var resultado;
// llamada y recogida de resultado devuelto
resultado = media(a1, a2, a3);
document.write(resultado);
```

### Funciones como argumentos

Una función puede recibir como parámetro otra función para que le ayude a hacer parte de su tarea.

*Ejemplo:* En el siguiente código la función *ahora()* devuelve la hora actual. La función *mostrar()* recibe una función como parámetro y muestra por pantalla lo que devuelva.

```
// Funcion que devuelve el momento actual
var ahora = function () {
    return Date();
}
// funcion que muestra lo devuelto por la funcion indicada
var mostrar = function (fn) {
    alert("Hoy es: " + fn());
}
// llamada a la funcion mostrar indicando la funcion ahora como argumento.
mostrar(ahora);
```

Al indicar la función *ahora()* como argumento para la función *mostrar()*, ésta muestra por pantalla la fecha actual.

### Funciones anónimas

También es posible declarar la función que se pasa como argumento en la propia llamada a la función como una función *anónimas*:

```
// llamada a la funcion mostrar indicando una funcion anónima como argumento
mostrar(function () {
    return Date();
});
```

### Funciones anónimas autoejecutables

Las funciones convencionales sólo se ejecutan cuando son llamadas. Una función anónima autoejecutable es aquella que se declara sin nombre para ser ejecutada en el momento sin llamada:

```
var func = function () {
    console.log("funcion normal llamada.")
}
// Llamada a funcion
func();

// LLamada a función anónima autoejecutable.
(function () {
    console.log("Funcion anónima autoejecutable");
})();
```

funcion normal llamada.

[HtmlPage1.html:10](#)

Funcion anónima autoejecutable

[HtmlPage1.html:17](#)

(\*) Una función anónima autoejecutable es definir una función de manera anónima como parte su propia llamada, con lo que se ejecuta en el acto y no puede ser llamada más tarde ( no tiene nombre ).

## Ambito de variables

### Concepto de ámbito

Se denomina “*ámbito*” de una variable al espacio en el código en el que es accesible y puede utilizarse.

### Globales

Las variables declaradas fuera de las funciones son accesibles desde dentro y fuera de las funciones:

```
var vble = 'hola';

var func = function () {
  console.log("Dentro funcion: " + vble); // OK
};

func();
console.log("Fuera funcion: " + vble); // OK
```

Dentro funcion: hola

HtmlPage1.html:12

Fuera funcion: hola

HtmlPage1.html:16

### Locales

Las variables declaradas dentro de una función no son accesibles fuera de la función.

```
var func = function () {
  var vble = 'hola';
  console.log("Dentro funcion: " + vble); // OK
};

func();
console.log("Fuera funcion: " + vble); // Error. Vble no es accesible:
```

Dentro funcion: hola

HtmlPage1.html:13

✖ Uncaught ReferenceError: vble is not defined

HtmlPage1.html:17

**Excepción:** Si la variable se declara dentro de la función sin poner delante “*var*”, se global y es accesible dentro y fuera de la función.

```
var foo = function () {
  mensaje = "HOLA";
  console.log("Dentro funcion: " + mensaje); // OK
}

foo();
console.log("Fuera funcion: " + mensaje); // OK
```

Dentro funcion: HOLA

HtmlPage1.html:11

Fuera funcion: HOLA

HtmlPage1.html:15

Dos variables pueden tener el mismo nombre perteneciendo a ámbitos distintos:

```
var vble = 'hola';

var func = function () {
  var vble = "adios"
  console.log("Dentro funcion: " + vble); // muestra vble local
};

func();
console.log("Fuera funcion: " + vble); // muestra vble global
```

Dentro funcion: adios

[HtmlPage1.html:13](#)

Fuera funcion: hola

[HtmlPage1.html:17](#)

### Ejercicio propuestos

1. Crea una función *esPar()* que recibe un valor como parámetro y retorna un valor lógico indicando si es o no par.
2. Crea una función *enumerador()* que reciba como parámetros dos valores y muestre en pantalla los valores comprendidos entre ambos de menor a mayor.
3. Crea una función *calcular()* que recibe como parámetros tres valores: a, b, y op, y devuelve un valor en base al valor del parámetro op:
  - a. Si op = 1 → Retorna la suma ( a + b )
  - b. Si op = 2 → Retorna la resta ( a - b )
  - c. Si op = 3 → Retorna el producto ( a x b )
  - d. Si op = 4 → Retorna el cociente ( a / b )
  - e. Si op no vale ninguno de los valores anteriores la función devuelve como resultado el mensaje "Operación no reconocida."
4. Crea una función *mostrarTabla()* que recibe como parámetro un valor y muestra en la tabla la tabla de multiplicar del valor correspondiente empleando *document.write()*.
5. Modifica la función del ejercicio anterior de modo que requiera dos parámetros que determinan un rango de valores válidos admitidos. La función solicitará al usuario un valor correcto hasta que introduzca un valor numérico comprendido en el rango indicado.