

Ordinogramas y Pseudocódigo

Algoritmo

Un algoritmo establece, de manera genérica e informal, la secuencia de pasos o acciones que resuelven un determinado problema.

También se puede decir que un algoritmo es la fase preliminar a escribir un programa en cualquier lenguaje de programación, por lo que la forma de escribirlo puede ser muy personal según el programador que lo esté diseñando, pero en general se debe hacer en una forma tan clara que cualquier otro programador pueda coger dicho algoritmo, entenderlo fácilmente y poderlo traducir a su lenguaje de programación favorito.

- Para conseguir este objetivo, se utilizan las siguientes notaciones **en programación**:

- pseudocódigo
- diagramas de flujo (ordinograma)

Ejemplo: Que debemos hacer si nos encontramos con una bombilla fundida:

[1] Comprobar si hay bombillas de repuesto

(1.1) Abrir el cajón de las bombillas

(1.2) Observar si hay bombillas

[2] Si hay bombillas:

(2.1) Coger la bombilla

(2.2) Coger una silla

(2.3) Subirse a la silla

(2.4) Poner la bombilla en la lámpara

[3] Si no hay bombillas

(3.1) Abrir la puerta

(3.2) Bajar las escaleras....

Características:

- Un algoritmo debe resolver el problema para el que fue formulado.
- Los algoritmos son independientes del ordenador.

- Los algoritmos deben de ser precisos.
- Los algoritmos deben de ser finitos.
- Los algoritmos deben de poder repetirse.

Elementos que conforman un algoritmo

- Entrada. Los datos iniciales que posee el algoritmo antes de ejecutarse.
- Proceso. Acciones que lleva a cabo el algoritmo.
- Salida. Datos que obtiene finalmente el algoritmo.

ORDINOGRAMAS

Los ordinogramas, también conocidos como diagramas de flujo, son una representación gráfica paso a paso de las instrucciones de un programa, reflejando la secuencia lógica de las operaciones para la resolución del problema.

Muestran gráficamente el algoritmo de un programa.

1. Debe tener un inicio y un fin.
2. Deben usarse líneas rectas.
3. Debe diseñarse de arriba abajo y de izquierda a derecha.
4. Debe guardarse la mayor simetría gráfica posible.
5. Deben usarse expresiones independientes de los lenguajes de programación.
6. Deben intentar evitarse los cruces de líneas.
7. Debe utilizarse el mínimo número de instrucciones posible.
8. Usar los mínimos comentarios posibles.

ORDINOGRAMAS

Están compuestos por los siguientes símbolos:

Inicio/fin del programa



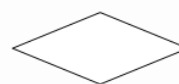
Subprograma



Operación general



Decisión (1 entrada, 2 ó 3 salidas)



Operación de E/S



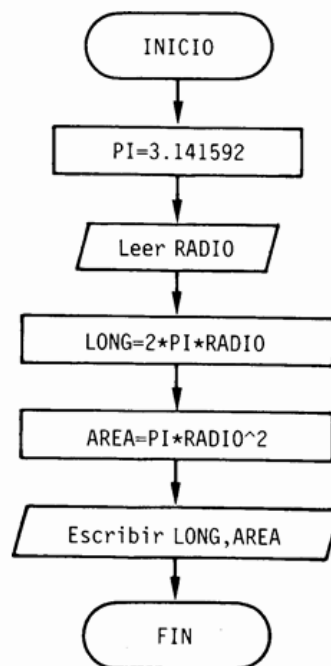
Conector



ORDINOGRAMAS

Ejemplo 1:

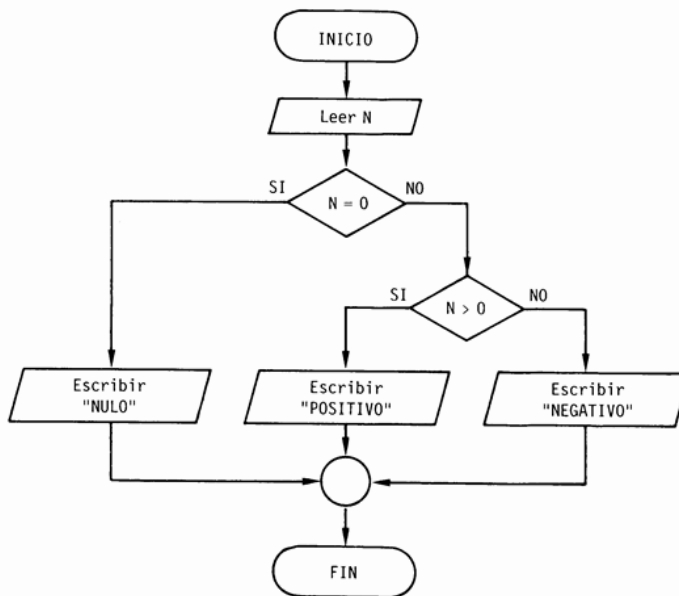
Programa que lee un número que corresponde con el radio de una circunferencia, calcula su longitud, su superficie y presenta el resultado



ORDINOGRAMAS

Ejemplo 2:

Lee un número de un dispositivo de entrada, comprueba e imprime en un dispositivo de salida estándar si el número es nulo, positivo o negativo

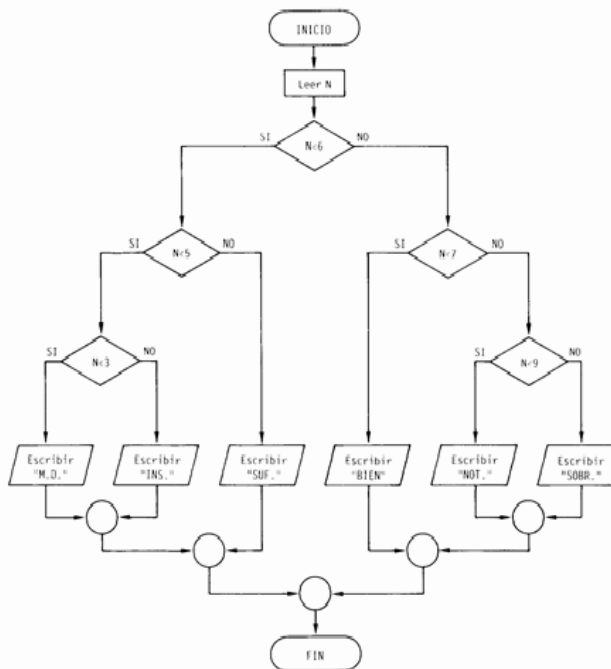


ORDINOGRAMAS

Ejemplo 3:

Programa que lee una calificación entre 0 y 10, y la transforma en una nota alfabética según la tabla:

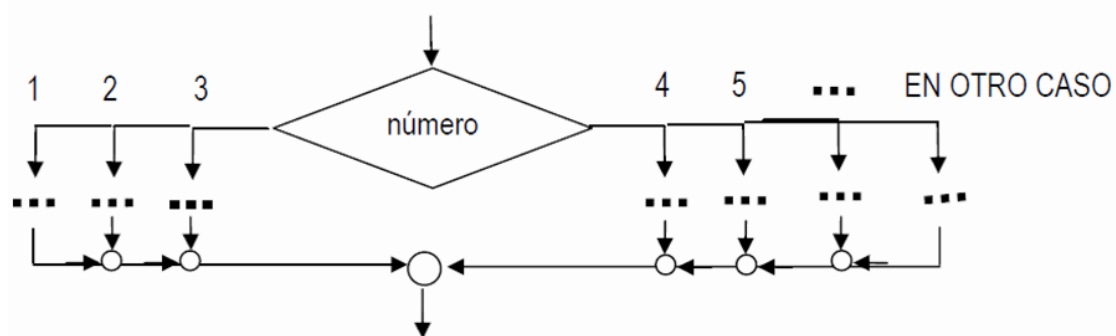
| Nota numérica | Nota alfabética |
|---------------|-----------------|
| 0 ≤ NOTA < 3 | M.D. |
| 3 ≤ NOTA < 5 | INS. |
| 5 ≤ NOTA < 6 | SUF. |
| 6 ≤ NOTA < 7 | BIEN |
| 7 ≤ NOTA < 9 | NOT. |
| 9 ≤ NOTA ≤ 10 | SOBR. |



ORDINOGRAMAS

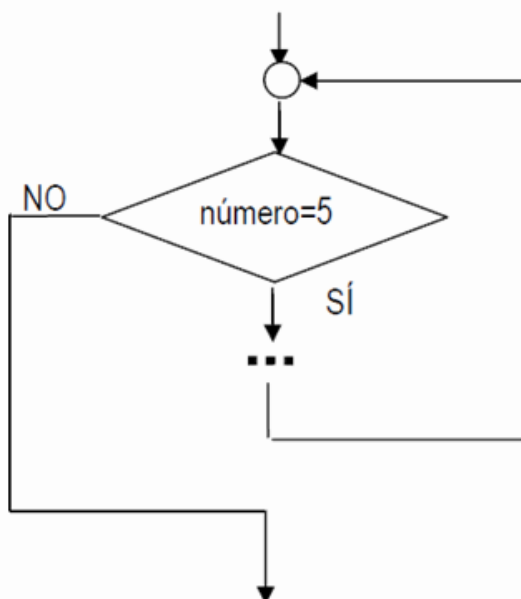
Existen más tipos de símbolos para situaciones concretas:

Evaluar

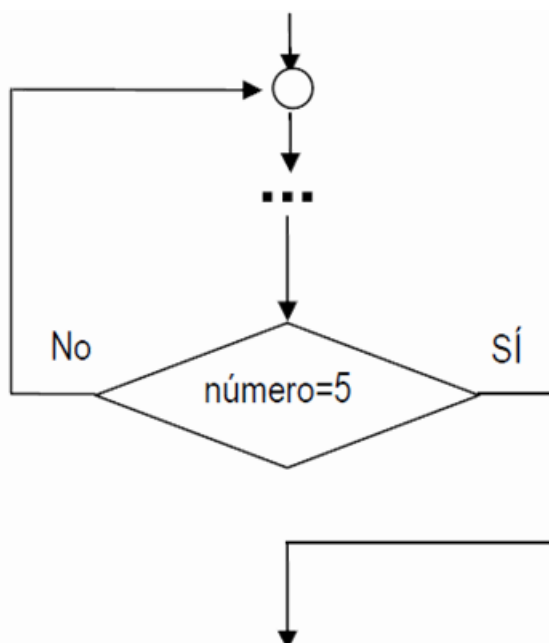


Existen más tipos de símbolos para situaciones concretas:

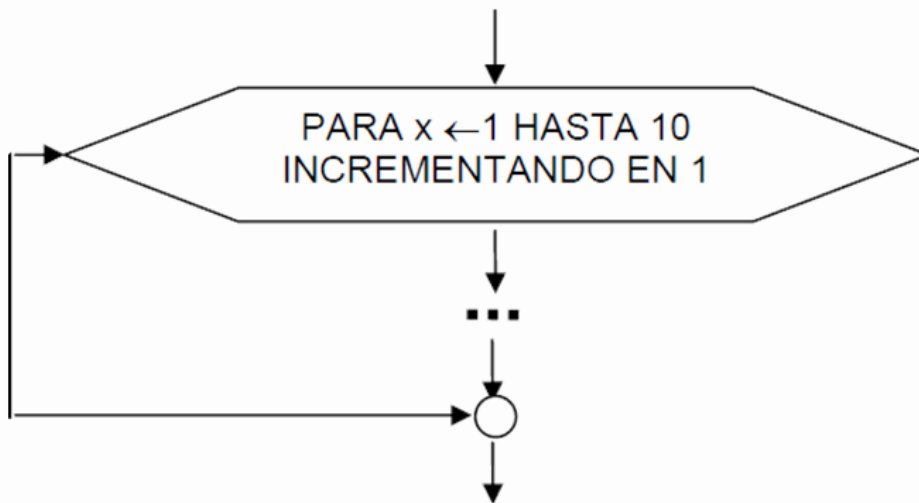
Bucle mientras:



Bucle hasta:



Bucle para:



PSEUDOCÓDIGO

Es una notación en la cual se escribe la solución al problema en forma de algoritmo (pasos ordenados), usando para ellos palabras y frases del lenguaje natural, pero siguiendo unas reglas determinadas:

- **Instrucciones primitivas:** son aquellas que hacen una operación, como por ejemplo asignar un valor a una variable.
- **Entrada de datos:** de teclado, fichero, etc.
- **Salida de datos:** pantalla, impresora, etc.
- **Instrucciones control de flujo:** dependiendo de la evaluación de una condición realizar una acción u otra.
- **Instrucciones de evaluación:** en función de uno de los múltiples valores que se pueden evaluar realizar la acción correspondiente.
- **Instrucciones de bucle:** para ejecutar repetidamente un bloque de instrucciones mientras se cumpla la condición.

Sintaxis en pseudocódigo de la instrucción de control de flujo **SI**:

SI se cumple condición ENTONCES

ejecutar instrucción o instrucciones

FIN SI

O

SI se cumple condición ENTONCES

Ejecutar instrucción o instrucciones

SI NO

Ejecutar otra instrucción o instrucciones

FIN SI

Sintaxis en pseudocódigo para la instrucción **EVALUAR**:

EVALUAR variable

SI VALE n:

ejecutar instrucción o instrucciones

SI VALE m:

ejecutar instrucción o instrucciones

:

:

SI VALE z:

ejecutar instrucción o instrucciones

EN OTRO CASO:

ejecutar instrucción o instrucciones

FIN EVALUAR

Sintaxis en pseudocódigo de las instrucciones de bucle **MIENTRAS, HACER y REPETIR**:

MIENTRAS se cumpla condición **HACER**

ejecutar instrucción o instrucciones

FIN MIENTRAS

HACER

ejecutar instrucción o instrucciones

HASTA QUE se cumpla condición

REPETIR

ejecutar instrucción o instrucciones

HASTA QUE se cumpla la condición

Sintaxis en pseudocódigo para la instrucción de bucle **PARA**:

PARA variable = inicio HASTA fin INCREMENTANDO EN tanto HACER
 ejecutar instrucción o instrucciones

FIN PARA

Donde:

- **inicio**: es el valor inicial de la iteración
- **fin**: es el valor final de la iteración
- **tanto**: en cuánto se incrementa la variable desde **inicio** hasta **fin** (normalmente vale 1)

Ejemplo: **PARA** n = 1 HASTA 5 INCREMENTANDO EN 1 HACER

XXXXXX

FIN PARA

Existen una serie de convenciones:

- Las palabras reservadas se escriben en mayúsculas: MOSTRAR, LEER, ESCRIBIR, SI, INICIO, FIN, etc
- Las variables en minúsculas o la primera letra en mayúsculas: palabra, Nombre.
- Textos encerrados en comillas dobles (""): "Hola Mundo", "Introduzca un valor:".
- Números sin comillas: 1, 34, -23, 3.1415, etc
- Letras o caracteres sueltos encerrados en comillas simples ('): 'a', '@', '#', etc.
- *Indentación* (tabulación) del código para mejorar la legibilidad.

Introducción JavaScript

1-Introducción.

JavaScript es considerado un lenguaje de programación interpretado, lo cual significa que se utiliza un intérprete para elaborarlo. Este intérprete convierte este lenguaje que codificamos a lenguaje máquina.

También se define como un lenguaje orientado a objetos, basado en prototipos y dinámico.

JavaScript se utiliza principalmente en entorno cliente, implementado como parte de un navegador web, permitiendo mejoras en la interfaz de usuarios y páginas web dinámicas, a pesar de que existe una parte de JavaScript ubicada en el lado del servidor.

Su uso en aplicaciones externas a la web, como por ejemplos en documentos PDF, aplicaciones de escritorio (sobre todo en “widgets”) es también significativo.

2-Lenguaje basado en guiones.

Un lenguaje de programación con guiones es una forma de lenguaje de programación que se suele interpretar en vez de compilar.

Los programas convencionales se convierten permanentemente en los archivos ejecutables antes de que se ejecuten, sin embargo, los lenguajes basados en guiones se interpretan en el momento de ejecutar la orden, una a una, cada vez que se ejecuta.

Estos lenguajes tienen como principal finalidad la de mejorar la gestión de una aplicación web, siendo ejecutados en el navegador del usuario, es decir, la parte llamada cliente.

Técnicamente, es más fácil escribir el código en un lenguaje de scripting que en uno compilado, no obstante, los lenguajes de de guión son mas lentos puesto que las instrucciones no son manejadas exclusivamente por el procesador de instrucciones básicas.

3-Ventajas e inconvenientes

- VENTAJAS

- a) El lenguaje de scripting es fácil de leer e interpretar.
- b) El código JavaScript se ejecuta en el cliente de manera que el servidor no es solicitado más de lo necesario, de esta manera se aligeran las tareas que ha de hacer el servidor.
- c) Son lenguajes que no necesitan ser compilados cada vez que se realiza una modificación, facilitando las pruebas en el momento de desarrollar la aplicación.
- d) Permiten el desarrollo del código y la aplicación de manera más rápida y pueden comunicarse fácilmente con los programas escritos en otros idiomas.

- INCONVENIENTES

- a) Los scripts tienen capacidades limitadas, por razones de seguridad, por lo cual no es posible hacerlo todo con JavaScript, sino que se ha de utilizar conjunto con otros lenguajes evolucionados, posiblemente más seguros, como PHP, HTML...
- b) El código es visible y puede ser leído por cualquiera, incluso si está protegido con leyes de copyright.
- c) El código del script se ha de descargar completamente antes de poder ejecutarse y si los datos que un script utiliza son muy elevadas el tiempo que tardará en descargarse será muy alto.

4- Creación de programas en JavaScript

El código JavaScript es ejecutado por el navegador cuando este es leído. En el caso de las funciones, estas han de ser llamadas desde alguna línea de código o desde los eventos que se generan en las etiquetas HTML.

El código JavaScript se incorpora al código HTML de una manera muy sencilla.

Para que el navegador reconozca que un archivo contiene funciones escritas en JavaScript estas funciones han de estar colocadas entre estas dos etiquetas:

```
<script language="javascript" type="text/javascript">  
..... codigo JavaScript  
</script>
```

La etiqueta script es una extensión de HTML en la cual se cierra el texto que compone el código del programa JavaScript correspondiente. Un documento puede tener varias etiquetas SCRIPT, y cada una de ellas incluirá sentencias JavaScript diferentes.

La etiqueta SCRIPT admite un parámetro opcional LANGUAGE que indica el lenguaje del script que se ha incrustado en el documento así como la versión de JavaScript.

Si el atributo LANGUAGE se omite, los navegadores asumen que es la última versión que ellos interpretan.

Importante! JavaScript es un lenguaje **Case Sensitive**, es decir, distingue entre mayúsculas y minúsculas, por lo tanto, hay que tener especial cuidado con la forma de escribir las variables y las órdenes del lenguaje.

Destacar, por último, que dentro de las etiquetas SCRIPT se pueden introducir comentario. Los de una sola línea empiezan con // mientras que los de bloque empiezan con /* y acaban con */.

```
<script language="javascript" type="text/javascript">  
    //Comentario de una sola linea  
    /* Comentario de bloque en el caso de que se necesiten varias lineas u omitir  
       parte del codigo*/  
</script>
```

Elementos del lenguaje script

Asignaciones y operadores: expresiones.

1) Asignaciones: expresiones.

El signo igual (=) se utiliza en JavaScript para indicar la acción de asignar un valor. Es decir, una sentencia de JavaScript podría ser por ejemplo:

Edad=3;

Los operadores que se utilizan en JavaScript son los mismos que se utilizan en cualquier otro lenguaje de programación.

Una expresión en JavaScript es una cosa que se pueda leer como una expresión booleana o una expresión numérica. Las expresiones contienen caracteres como “+” en lugar de expresarlo como “sumado a”. Cualquier combinación válida de valores, variables, operadores y expresiones constituyen una expresión.

Ejemplos de expresiones:

var Expresion1 = 3 * (4 / 5);
var Expresion2 = “Hola” + “ que tal”;
var Expresion3 = 4+6;
var Expresion4 = Expresion1 + Expresion3;

2) Operadores

Combinando variables y valores, se pueden formular expresiones más complejas. Las expresiones son una parte fundamental de los programas. Para formular expresiones se utilizan los operadores.

Operadores aritméticos

| Operador | Descripción |
|----------|----------------------|
| + | Suma |
| - | Resta |
| * | Multiplicación |
| / | División |
| % | Resto de la división |

Operadores de asignaciones

| Operador | Descripción |
|-------------|--|
| = X=5; | Asigna a la variable de la parte izquierda el valor de la derecha. La variable obtiene el valor 5. |
| += X+=2; | Suma los operandos de la izquierda y la derecha y asigna el resultado al operando de la izquierda. La variable obtiene el valor de 7. |
| -= X-=3; | Resta los operandos de izquierda y derecha y asigna el resultado al operando de la izquierda. La variable obtiene el valor de 4. |
| *= X*=4; | Multiplica los operandos de izquierda y derecha y asigna el resultado al operando de la izquierda. La variable obtiene el valor de 16. |
| /= X/=2; | Divide los operandos de izquierda y derecha y asigna el resultado al operando de la izquierda. La variable obtiene el valor de 8. |
| %= X%=3; | Divide los dos operandos y asigna el resto de la división al operando de la izquierda. El resultado es 2 (la resta de 8/3 es 2). |

Operadores de comparación

| Operador | Descripción |
|----------|---|
| == | Devuelve el valor TRUE cuando los dos operandos son iguales. |
| != | Devuelve el valor FALSE cuando los dos operandos son diferentes. |
| > | Devuelve el valor TRUE cuando el valor de la izquierda es mayor que el de la derecha. |
| < | Devuelve el valor TRUE cuando el valor de la izquierda es menor que el de la derecha. |
| >= | Devuelve el valor TRUE cuando el valor de la izquierda es mayor o igual que el de la derecha. |
| <= | Devuelve el valor TRUE cuando el valor de la izquierda es menor o igual que el de la derecha. |

Operadores lógicos

| Operador | Descripción |
|----------|---|
| && | I lógica. Devuelve el valor TRUE cuando los dos operandos son TRUE. |
| | O lógica. Devuelve el valor TRUE cuando uno de los dos operandos es TRUE. |
| ! | Negación lógica. Devuelve el valor TRUE cuando el valor es falso. |

Variables. Tipos y ámbitos. Conversiones.

Las variables son elementos del lenguaje que permiten almacenar diferentes 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.

Por ejemplo, podemos solicitar el nombre del usuario al entrar en una web y almacenarlo en una variable para posteriormente, cuando el usuario finalice la sesión despedirlo con una frase que incluya su nombre.

1) Nombre de las variables

Cada variable se identifica por un nombre. Al asignar un nombre a una variable hay que tener en cuenta estas reglas:

- Se pueden utilizar todas las letras del abecedario en mayúsculas y minúsculas, los números del 0 al 9 y el guion subrayado “_”.
- Los nombres de las variables no pueden contener puntos ni espacios en blanco.
- El primer carácter deber ser una letra o el guion subrayado.
- En el nombre se distinguen mayúsculas y minúsculas.
- No se pueden utilizar como nombre ninguna de las palabras reservadas por JavaScript, es decir, no se pueden utilizar palabras que coincidan con las que pertenecen al lenguaje JavaScript.
- Oficialmente no hay restricciones en lo que respeta a la longitud del nombre, pero esta ha de caber en una sola línea.

| Nombres correctos | Nombres incorrectos |
|-------------------|---------------------|
| Aux_Nombre | Aux Nombre |
| resultado | primer.resultado |
| mail | 2mail |
| primer_apellido | document |

2) Declaración de variables

Para declarar variables en JavaScript se utiliza la instrucción **var**. A cada variable se le asigna un nombre y opcionalmente un valor inicial. Si no se trata de una función, la instrucción **var** es opcional, pero se recomienda utilizarla.

var ejemplo;

Hola = "Hola"; //es lo mismo que var Hola = "Hola";

var resultat=3+7; //la variable tendrá el valor de 10

3) Tipos de variables

Los tipos de valores que puede contener una variable JavaScript son:

- **Números:** Pueden contener cualquier tipo de número, real o entero.
- **Operadores lógicos o booleanos:** Puede contener uno de los siguientes valores: TRUE, FALSE, 1 o 0;
- **Cadenas de caracteres o String:** Cualquier combinación de caracteres (letras, números, signos especiales y espacios). Las cadenas se delimitan mediante comillas dobles o simples. Las comillas simples, por norma, serán utilizadas dentro de fragmentos de código delimitados por comillas dobles o viceversa.

Ejemplos:

| Tipos de variable | Variable |
|-------------------|---|
| Númerica | numero1=3; numero2 = 3.7; numero3 = 0.6; numero4 = 5+7; |
| String | cadena1 = "Pepe"; cadena2 = "Bienvenido al site web"; |
| Booleana | entrada = true; salida = false; auxentrada = 1; auxsalida = 0; |

En JavaScript existen también las matrices o vectores (**Arrays**), que son grupos de elementos que pueden ser de diferentes tipos. Se pueden declarar de diversas formas:

Usando el *constructor* **Array()**:

```
var coches = new Array();
```

```
coches[0] = "Volvo";
```

```
coches[1] = "Seat";
```

```
coches[2] = "Renault";
```

De forma condensada: `var coches = new Array("Volvo", "Seat", "Renault");`

De forma *literal*: `var coches = ["Volvo", "Seat", "Renault"];`

Las matrices pueden ser de diferentes *dimensiones*, donde los elementos se enumeran de 0 a $n-1$, donde n es el número total de elementos de la matriz:

- Una dimensión:

```
var coches = new Array("Volvo", "Seat", "Renault");
```

- Dos dimensiones:

```
var personas = new Array(new Array("John", "Doe"), new Array("Will", "Smith"), new Array("John", "Smith"))
```

// o también de la forma:

```
var personas= [["John", "Doe"], ["Will", "Smith"], ["John", "Smith"]]
```

// Obtener el dato de la fila 1 columna 0

```
var persona = personas[1][0]; // asigna el valor "Will"
```

4) Palabras reservadas en JavaScript

JavaScript tiene una serie de palabras reservadas que no pueden ser utilizadas como nombre de variables:

| | | | | |
|----------|----------|--------|--------|-------|
| Break | False | In | This | Void |
| Continue | For | New | True | While |
| Declare | Function | Null | TypeOf | With |
| Else | If | Return | Var | |

5) Caracteres especiales en JavaScript

JavaScript tiene también la opción de introducir caracteres especiales (acentos, <, >, " ...). Para hacerlo se debe utilizar una codificación especial, en el caso de querer introducir caracteres especiales dentro del código HTML se ha de hacer:

\u00e1 -> á
\u00e9 -> é
\u00ed -> í
\u00f3 -> ó
\u00fa -> ú
\u00c1 -> Á
\u00c9 -> É
\u00cd -> Í
\u00d3 -> Ó
\u00da -> Ú
\u00f1 -> ñ
\u00d1 -> Ñ

6) Integración del código JavaScript

Como ya se ha comentado en apartados anteriores, para incluir un código JavaScript se utilizan las etiquetas `<script></script>` en el lugar del documento HTML que sea necesario.

Ejemplo:

```
<HTML>
  <HEAD>
  </HEAD>
  <BODY>
    <SCRIPT LANGUAGE="Javascript">
      var ciudad="Oviedo";
      var edad=6;carnet=true;
    </SCRIPT>
  </BODY>
</HTML>
```

Incluir el código JavaScript en medio del código HTML puede dar varios problemas y es recomendable hacerlo solo si no hay otro remedio. La otra opción de incluir el código es creando ficheros con la extensión ".js" donde se incluirá el código JavaScript y después se hará referencia a estos ficheros en el documento principal HTML; estas referencias se han de incluir en la cabecera del documento principal `<head>`.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Control asistencia</title>
  <script src="../jscalendar/calendar.js" type="text/javascript" xml:space="preserve">
  </script>
  <script src="../jscalendar/lang/calendar-ca.js" type="text/javascript"
xml:space="preserve">
  </script>
  <script src="../jscalendar/calendar-setup.js" type="text/javascript"
xml:space="preserve">
  </script>
</head>
<body>
</body>
</html>
```

7) Rutas relativas y absolutas

A la hora de incluir un fichero JavaScript se debe indicar el lugar donde se encuentra este fichero. Dentro de la etiqueta `<script>` encontramos un parámetro que es **SRC** donde se indica precisamente esta información.

Las rutas que se incluyen en este parámetro pueden ser relativas o absolutas, se escogerá una opción en función de la aplicación que se está diseñando, aunque se aconseja utilizar siempre rutas relativas, de esta manera, si la aplicación se ha de migrar a otro servidor donde hay variación de rutas, no se deberá modificar ninguna ruta porque la página web, por defecto, cogerá las nuevas rutas.

Ejemplo de ruta relativa:

```
<script src="../../jscalendar/calendar.js" type="text/javascript"
xml:space="preserve" />
```

En este caso el fichero “calendar.js” al que hace referencia esta parte del código se encuentra en la carpeta “jscalendar” la cual cuelga de otra carpeta que será la raíz donde se encuentra la página web.

Ejemplo de ruta absoluta:

```
<script src="/var/www/intraproven//jscalendar/calendar.js" type="text/javascript"
xml:space="preserve" />
```

En este caso se ha incluido la ruta absoluta, el fichero “calendar.js” se encuentra directamente en “/var/www/intraproven//jscalendar/calendar.js” y en el caso de que se realice una migración, la aplicación debería ser modificada para incluir las nuevas rutas

Estructura de control IF

Introducción

JavaScript ofrece diferentes posibilidades de romper la estructura lineal y secuencial de la ejecución de las líneas de instrucción.

Con la ayuda de las ramificaciones se pueden comprobar determinadas condiciones y, de esta manera, ejecutar una u otra orden. Los bucles nos permiten repetir un conjunto de instrucciones hasta que se cumpla una condición concreta.

Estructura IF

Las estructuras de decisión se utilizan para realizar una u otra acción en función de la condición o condiciones que evalúen:

SI (edad >= 18) // Si mayor o igual a 18 años realizar una acción

{

Acciones a realizar.

}

SINO // si no lo es realizar otra acción

{

Acciones a realizar.

}

La estructura de decisión **if** tiene la siguiente sintaxis básica:

if (expresión)

{

// Acción o acciones a realizar

// ...

}

else

{

// Acción o acciones alternativas

// ...

}

En caso de que *expresión* al ser evaluada sea cierta (**true**) se ejecutará lo que se encuentre delimitado por las primeras llaves, en caso contrario, se ejecutará lo que esté en las segundas.

Se pueden anidar tantas sentencias **if-else** como sea necesario, siempre y cuando se tenga especial cuidado en su anidación:

if (expresión1)

{

 if (expresión2)

 {

 // Acción 1

 }

 else

 {

 // Acción 2

 }

}

else

{

 if (expresión3)

 {

 // Acción 3

 }

 else

 {

 // Acción 4

 }

}

Ejemplo:

```
if(habitantesCiudad1 > habitantesCiudad2)
{
    alert("La ciudad "+nombreCiudad1+" tiene más habitantes que la ciudad "+nombreCiudad2);
}
else
{
    if (habitantesCiudad1 < habitantesCiudad2)
    {
        alert("La ciudad "+nombreCiudad1+" tiene menos habitantes que la ciudad "+nombreCiudad2);
    }
    else
    {
        alert("Las ciudades "+nombreCiudad2+" y "+nombreCiudad1+" tienen el mismo numero de habitantes.");
    }
}
```

Otra manera de ejecutar el ejemplo anterior:

```
if(habitantesCiudad1 > habitantesCiudad2)
{
    alert("La ciudad "+nombreCiudad1+" tiene más habitantes que la ciudad "+nombreCiudad2);
}
else if (habitantesCiudad1 < habitantesCiudad2)
{
    alert("La ciudad "+nombreCiudad1+" tiene menos habitantes que la ciudad "+nombreCiudad2);
}
else
{
    alert("Las ciudades "+nombreCiudad2+" y "+nombreCiudad1+" tienen el mismo numero de habitantes.");
}
```

EJERCICIO DNI:

Tabla para Calcular el DNI NIF

| | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 |
| T | R | W | A | G | M | Y | F | P | D | X | B | N | J | Z | S | Q | V | H | L | C | K | E |

Estructura de control WHILE

Estructura WHILE

Con el bucle **WHILE** se pueden ejecutar un grupo de instrucciones mientras se cumpla una condición determinada. Si la condición nunca se cumple, entonces tampoco se ejecutará ninguna instrucción.

Si la condición se cumple siempre, nos veremos inmersos dentro del problema de los bucles infinitos, que pueden llegar a colapsar el navegador, o incluso, el propio ordenador.

Por esta razón es muy importante que la condición deje de cumplirse en algún momento.

Sintaxis

```
while (Condición)
{
    ...instrucciones...
}
```

Ejemplo

Mostrar por pantalla los 60 primeros números pares:

```
var i = 1;

while(i<30)
{
    alert(i*2);
    i++;
}
```

Estructura DO WHILE

La diferencia del bucle **DO WHILE** delante del bucle **WHILE** reside en el momento en que se comprueba la condición: el bucle **DO WHILE** no la comprueba hasta el final, es decir, después del cuerpo del bucle, lo que significa que el bucle **DO WHILE** se recorrerá, mínimo, una vez, aun qué no se cumpla la condición.

Sintaxis

```
do
{
    ...instrucciones...
}
while (Condición)
```

Ejemplo

Pide de introducir la clave correcta. Mientras no sea anonimo se la seguirá pidiendo.

```
var claveAux="";

do
{

    claveAux =prompt("Introduce la clave ","anonimo");

}

while (claveAux != "anonimo");

alert("Has acertado la clave");
```

Parada de un bucle: Instrucciones BREAK

En los bucles **FOR** y **WHILE** se pueden utilizar las instrucciones **BREAK** y **CONTINUE** para modificar el comportamiento del bucle. La instrucción **BREAK** dentro del bucle hace que este se pare inmediatamente, aunque no se haya ejecutado el bucle completamente.

Al llegar a la instrucción, el programa se sigue desarrollando inmediatamente a continuación del bucle.

Ejemplo

Se pide la contraseña al usuario. Si falla tres veces sale del bucle

```
var auxClave="";  
  
var numVeces=0;  
  
while (auxClave!= "anonimo")  
{  
  
    auxClave=prompt("Introduce la clave ", "");  
  
    numVeces ++;  
  
    if (numVeces == 3)  
    {  
  
        break;  
  
    }  
  
}  
  
if (auxClave == "anonimo")  
{  
  
    alert("La clave es correcta");  
  
}  
  
else  
{  
  
    alert("La clave es incorrecta");  
  
}
```

Parada de un bucle: Instrucciones CONTINUE

El efecto que tiene la instrucción **CONTINUE** en un bucle es la de retornar a hacer la secuencia de ejecución a la cabecera del bucle, volviendo a ejecutar la condición o incrementar los índices cuando sea un bucle **FOR**.

Esto permite saltar recorridos del bucle.

Ejemplo

Mostrar por pantalla los números pares hasta el 60 excepto el 40.

```
var i = 0;
while(i<30)
{
    i++;
    if(i==20)
    {
        continue;
    }
    alert(i*2);
}
```

Estructura de control FOR

Bucle

Muchas veces, como en el caso de la estructura **WHILE**, es necesario que un grupo de ordenes se repita varias veces hasta que se cumplan unas determinadas condiciones.

De realizar este tipo de secuencia de ejecución se utilizan unas estructuras llamadas bucles.

Estructura/Bucle FOR

Cuando la ejecución de un programa llega a un bucle **FOR**, lo primero que se hace es ejecutar la inicialización del índice y después analiza la condición de prueba, si esta se cumple se ejecutan las instrucciones que contiene el bucle.

Al llegar al final del bucle se realiza la modificación del índice y de nuevo se analiza la condición de prueba; si esta se sigue cumpliendo se ejecutan las instrucciones del bucle y así sucesivamente hasta que la condición de prueba no se cumplan.

Una vez llegado a este punto, el programa salta a las instrucciones que hay después del bucle.

Sintaxis

for (inicialización; expresión; actualización)

```
{  
    // Acción o acciones a realizar  
    // ....  
}
```

En **inicialización** por ejemplo se indica el valor de inicio de la variable de contador.

En **expresión** la condición que se ha de cumplir para que el bucle continúe.

En **actualización** se indica qué acción se realiza con la variable *inicialización*.

Ejemplo

Bucle **FOR** que escribe en pantalla los números del 0 al 9:

```
var i;
```

```
for (i = 0; i < 10; i++)
```

```
{  
    document.writeln(i);  
}
```

Se *inicializa* el valor de la variable *i* a 0.

El bucle se va ejecutando mientras la *expresión* *i < 10* se cumpla.

Después de escribir el número se *actualiza* el valor de la variable incrementándola en 1.

En el bucle **FOR** la variable utilizada como *contador* se puede incrementar en valores mayores de 1:

```
var i;  
  
for (i = 0; i < 33; i += 3)  
{  
    document.writeln(i);  
}
```

En éste ejemplo el *contador* se *actualiza* al final del bucle sumándole 3 en cada iteración.

También se puede *actualizar* la variable que hace de *contador* decrementando su valor, ya sea en 1 u otros valores mayores de 1:

```
var i;  
  
for (i = 10; i > 0; i--)  
{  
    document.writeln(i);  
}
```

Por último indicar que la declaración de la variable *contador* también se puede hacer dentro de la parte de *inicialización* del bucle **FOR**. Es la forma más común de declaración.

```
for (var i = 33; i > 0; i -= 3)
{
    document.writeln(i);
}
```

Estructura de control SWITCH

Estructura SWITCH

La estructura de control **SWITCH** permite realizar múltiples comparaciones de valores, que si alguna se cumple ejecutará una acción, y opcionalmente se puede definir una acción por defecto si no se cumple ninguna.

Sintaxis

switch (variable)

{

case expresion1: // si variable igual a expresion1

// Acción 1

break;

case expresionN: // si variable igual a expresionN

// Acción N

break;

default: // Si no se cumple ninguna de las expresiones anteriores

// Acción por defecto

}

Ejemplo

Estructura switch que dependiendo de que nombre se le pasa se le indica un saludo junto a ese nombre:

```
switch (nombre)
{
    case "Alberto":
        alert ("Hola Alberto");
        break;

    case "Carmen":
        alert ("Hola Carmen");
        break;

    default:
        alert ("¿quién eres tu?");
        break;
}
```

Entrada-Salida | TRY-CATCH

Entrada y salida: Introducción:

Hablar de diálogos en JavaScript es hablar de la manera en la que JavaScript es capaz de comunicarse con el usuario, preguntándole si quiere realizar una determinada acción o alertándolo de que ha habido un error o que se ha producido alguna circunstancia en el momento de la ejecución de la página.

Alert(texto):

La función alert(texto) es un cuadro de alerta que se utiliza a menudo si uno quiere asegurarse que cierta información, de aviso o alerta, llegue al usuario.

Cuando aparece un cuadro de alerta, el usuario deberá hacer click en “Aceptar” para poder continuar.

Principalmente se utiliza para alertar al usuario que ha habido un error en la página, como por ejemplo en los casos de formularios, indicando que un campo es erróneo.

El parámetro texto de la función es el texto que se le mostrará al usuario.

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

    function show_alert()
    {
        alert("Cuidado, has pulsado el boton!");
    }

</script>
</head>
<body>
<input type="button" onclick="show_alert()" value="Haz click y veras." />
</body>
```

Confirm(texto):

Una casilla de confirmación es la función confirm(text), se utiliza muy a menudo si se quiere que el usuario verifique o acepte alguna cosa.

Cuando se muestra una casilla de confirmación, el usuario tendrá la opción de clicar en “Aceptar” o “Cancelar” para continuar. Si el usuario clicka en “De acuerdo”, la caja devuelve True, en caso contrario devolverá False.

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

    function show_confirm()
    {
        var respuesta=confirm("Estas de acuerdo?");

        if (respuesta)
        {
            alert("Estas de acuerdo!");
        }
        else
        {
            alert("No estas de acuerdo!");
        }
    }
</script>
</head>
<body>
<input type="button" onclick="show_confirm()" value="Confirma!" />
</body>
```

Prompt():

Un cuadro de mensaje `prompt(texto, valorPorDefecto)` se utiliza a menudo si se quiere que el usuario introduzca un valor antes de entrar en la página.

En el cuadro, el usuario deberá clickar en “Aceptar” o “Cancelar” para continuar después de entrar un valor de entrada. Si el usuario clicka en “Aceptar” el cuadro devuelve el valor introducido.

Si clicka en “Cancelar” la caja devuelve null.

```
<html>
<head>
<script type="text/javascript">
    function show_prompt()
    {
        var nombre=prompt("Dime tu nombre","Cristian");

        if (nombre!=null && nombre!="")
        {
            document.write("<p>Hola " + nom + "! Como estás?</p>");
        }
    }
</script>
```

Salida en ventana:

En JavaScript para escribir en la ventana del navegador existen dos métodos:

```
document.write("texto");
```

```
document.writeln("texto");
```

La diferencia entre las dos es que la segunda añade al texto a imprimir los caracteres para realizar un salto de línea en el código HTML de la página

Ejemplo:

```
var texto = "Hola Mundo";
```

```
document.writeln(texto); // Imprime el contenido con salto de línea
```

```
document.write("Texto impreso sin incluir salto de línea al final.");
```


Manejo de errores: TRY y CATCH:

Durante la ejecución de un código se pueden producir errores que provocan una excepción y su parada.

En las últimas versiones de JavaScript se ha adoptado el mecanismo de manejo de errores de otros lenguajes como C++, C#, Java y PHP, las sentencias **try** y **catch**.

Sintaxis:

```
try
{
    // Código con un posible error
}
catch (err) // En caso de producirse una excepción
{
    alert(err.message); // Presentar una alerta
}
```

Ejemplo:

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

    try
    {
        var numero = aa;
        document.writeln("El factorial de " + numero + " es " + factorial(numero));
    }
    catch (error)
    {
        alert("Se ha producido el error " + error.message + '!');
    }

    function factorial(numero)
    {
        var resultado = 1;
        for(var i=1; i<=numero; i++)
        {
            resultado *= i;
        }

        return resultado;
    }

</script>
</head>
<body>
</body>
```

En JavaScript también se puede provocar una excepción a través de la sentencia **throw**.

Ejemplo:

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

    try
    {
        var numero = -1;
        if(numero<1) throw ("Factorial de numero negativo!")
        document.writeln("El factorial de " + numero + " es " + factorial(numero));
    }
    catch (error)
    {
        alert(error);
    }

    function factorial(numero)
    {
        var resultado = 1;
        for(var i=1; i<=numero; i++)
        {
            resultado *= i;
        }

        return resultado;
    }

</script>
</head>
<body>
</body>
```

Documentación

En programación siempre es muy importante tener bien documentado el código por varios motivos:

- Permitir que el código sea comprensible para ojos ajenos.
- Recordar exactamente el funcionamiento de los métodos o funciones.
- Comprender las variables.

Comentarios:

En código **HTML** los comentarios se escriben entre `<!-- comentario -->`

En código **JavaScript** existen dos maneras:

- Si el comentario es para una línea sirve con poner `// comentario`
- Si el código contiene mas de una línea debe ir entre `/* comentario */`

Documentación:

Siempre es importante que cada script y cada función tenga una cabecera de documentación:

Esta seria la cabecera del script:

```
/*  
  
* @author: Cristian Garcia  
  
* @versio: 1.0  
  
* @description: this script controls all functions in index.html  
  
* @date: 2014/09/30  
  
*  
  
*/
```

Y esta seria la de las funciones:

/*

* @name: loadPage

* @author: Cristian Garcia

* @versio: 1.0

* @description: function to load some html content in index.html

* @date: 2014/09/30

* @params: none

* @return: none

*

*/

Funciones

Introducción a las funciones:

Cuando se desarrolla una aplicación compleja, es muy habitual utilizar una y otra vez las mismas instrucciones.

Por ejemplo, un script para una tienda de comercio electrónico ha de calcular el precio total de los productos varias veces, para añadir los impuestos y los gastos de envío.

Cuando una serie de instrucciones se repiten una y otra vez, se complica mucho el código fuente de la aplicación ya que:

- El código es mucho más largo porque muchas instrucciones estar repetidas.
- Si se quiere modificar alguna de las instrucciones repetidas, se ha de hacer tantas modificaciones como veces se haya escrito esta instrucción, lo que lo convierte en un trabajo pesado y muy propenso a cometer errores.

Las funciones son la solución a todos estos problemas, tanto en JavaScript, como en la resta de lenguajes de programación.

Una función es un conjunto de instrucciones que se agrupan para realizar una tarea concreta y que se pueden reutilizar fácilmente.

```
<script type="text/javascript">  
    var resultado;  
    var numero1 = 3;  
    var numero2 = 5;  
    resultado = numero1 + numero2;  
    alert("El resultado és " + resultado);  
  
    numero1 = 10;  
    numero2 = 7;  
    resultado = numero1 + numero2;  
    alert("El resultado és " + resultado);  
  
    numero1 = 5;  
    numero2 = 8;  
    resultado = numero1 + numero2;  
    alert("El resultado es " + resultado);  
</script>
```

En el ejemplo anterior, a pesar de lo absurdo que parece, resulta más absurdo aún repetir constantemente el código teniendo en cuenta que ejecutan lo mismo.

La solución que proponen las funciones consiste en extraer las instrucciones que se repiten y sustituirlas por una instrucción del tipo “en este punto se ejecutan las instrucciones que se han extraído”:

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

```
var resultat;  
var numero1 = 3;  
var numero2 = 5;
```

```
/* En este punto se le dice a la funcion que sume 2 numeros y muestre el resultado */  
/
```

```
numero1 = 10;  
numero2 = 7;
```

```
/* En este punto se le dice a la funcion que sume 2 numeros y muestre el resultado */  
/
```

```
numero1 = 5;  
numero2 = 8;
```

```
/* En este punto se le dice a la funcion que sume 2 numeros y muestre el resultado */  
/
```

```
</script>
```

Entonces, para que la solución anterior sea válida, las instrucciones comunes se deberían agrupar en una función a la cual se le pueda indicar los número que se han de sumar antes de mostrar el mensaje.

Por tanto, en primer lugar se ha de crear la función básica con las instrucciones comunes.

Las funciones en JavaScript se definen mediante la palabra reservada **function**, seguida del nombre de la función. Su definición formal seria la siguiente:

```
Function nombre_funcion(parámetros si los tiene)  
{  
    Instrucciones...  
}
```

El nombre de la función se utiliza para llamar a esa función cuando sea necesario. El concepto es el mismo que con las variables, a las cuales se les asigna un nombre único para poder utilizarlas dentro del código.

Después del nombre de la función, se incluirían dos paréntesis entre los cuales vendrían los argumentos que recibiría esa función. Más adelante se explicarán.

Finalmente, los símbolos `{ i }` se utilizan para cerrar todas las instrucciones que pertenecen a la función (de forma similar a como se cierran las instrucciones en las estructuras IF y FOR).

Volviendo al ejemplo anterior, se crea una función llamada **suma_y_muestra** de la manera siguiente:

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

```
function suma_y_muestra()
{
    resultado = numero1 + numero2;
    alert("El resultado es " + resultado);
}
```

```
</script>
```

El código final quedaria así:

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

```
function suma_y_muestra()
{
    resultat = numero1 + numero2;
    alert("El resultat és " + resultat);
}
```

```
var resultado;
var numero1 = 3;
var numero2 = 5;
suma_y_muestra();
```

```
numero1 = 10;
numero2 = 7;
suma_y_muestra();
```

```
numero1 = 5;
numero2 = 8;
suma_y_muestra();
```

```
</script>
```


El código del ejemplo anterior es mucho más eficiente que el primer código mostrado, ya que no existen instrucciones repetidas.

Las instrucciones que suman y muestran mensajes se han agrupado bajo una función, lo que permite ejecutarla en cualquier punto del programa simplemente indicando el nombre de la función.

No obstante, aún faltaría un pequeño pero importante dato para que este código quedase totalmente optimizado; los argumentos.

Argumentos.

Las funciones más sencillas no necesitan ninguna información para producir sus resultados.

No obstante, la mayoría de funciones de las aplicaciones reales han de acceder al valor de algunas variables para producir sus resultados.

Las variables que necesitan las funciones se llaman argumentos. Antes de poder utilizarlos, la función ha de indicar cuantos argumentos necesita y cuales son los nombres de cada argumento.

Además, al invocar la función, se han de incluir los valores que se le pasaran a la función.

Los argumentos se indican dentro de los paréntesis que van detrás del nombre de la función y se separan por comas.

Siguiendo el ejemplo de la suma de números, la función ha de indicar que necesita dos argumentos, correspondientes a los dos números que ha de sumar:

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

```
function suma_y_muestra(numero1,numero2)
{
    var resultado = numero1 + numero2;
    alert("El resultado es " + resultado);
}
```

Dentro de la funció, el valor de la variable numero1 serà igual al primer valor que se passe a funció y el valor de la variable numero2 es igual al segundo valor que se le pasa.

Para pasar valores a la funció, se incluyen dentro de los paréntesis utilizados al llamar a la funció:

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

    function suma_y_muestra(num1,num2)
    {
        var resultado = numero1 + numero2;
        alert("El resultado és " + resultado);
    }
    var numero1 = 3;
    var numero2 = 5;
    suma_y_muestra(numero1,numero2);

</script>
```

En el código anterior se ha de tener en cuenta que:

- Aunque casi siempre se utilizan variables para pasar los datos a la función, se podría haber utilizado directamente el valor de estas variables: **suma_y_muestra(3,5);**
- El número de argumentos que se pasan a una función deberían ser los mismo que el número de argumentos que se han indicado en la función. No obstante, JavaScript no muestra ningún error si se le pasan más o menos argumentos.
- El orden de los argumentos es fundamental, ya que el primer dato que se indica en la llamada será el primer valor que espere la función, el segundo que se indica en la llamada será el segundo que espere la función y así sucesivamente.
- Se puede utilizar un número ilimitado de argumentos, aunque contra más argumentos más se complica la llamada a la función.
- No es obligatorio que coincidan el nombre de los argumentos que utiliza la función y el nombre de los argumentos que se le pasan.

Valores de retorno de una función

Las funciones no solo pueden tener parámetros de entrada sino que también pueden devolver valores que se pueden utilizar más adelante en el documento.

Pongamos un ejemplo para tenerlo más claro. A continuación se muestra otro ejemplo de una función que calcula el precio total de un producto a partir de su precio básico:

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

// Definición de la función
function calculaPrecioTotal(precio)
{
    var impuestos = 1.16;
    var gastosEnvio = 10;
    var precioTotal = ( precio * impuestos ) + gastosEnvio;
}

// Llamada a la función
calculaPrecioTotal (23.34);

</script>
```

La función anterior toma como argumento una variable llamada precio y se le suma los impuestos y los gastos de envío para obtener el precio total. Al llamar a la función se le pasa directamente el valor del precio básico mediante el número 23,34.

No obstante el código, anterior no es muy útil puesto que lo ideal sería que la función pudiese devolver el resultado obtenido para guardarlo en otra variable y poder seguir trabajando con ella:

Afortunadamente, las funciones no solo pueden recibir variables y datos, sino que también pueden devolver los valores que han calculado. Para devolver los valores dentro de la función se utiliza la palabra clave **return**.

Aunque las funciones pueden devolver valores de cualquier tipo, solo pueden devolver uno cada vez que se ejecutan.

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

// Definición de la función
function calculaPrecioTotal(precio)
{
    var impuestos = 1.16;
    var gestosEnvio = 10;
    var precioTotal = (precio * impuestos) + gestosEnvio;
    return precioTotal;
}

// El valor retornat per la funció, es guarda en una variable
var precioTotal = calculaPrecioTotal (23.34);

// Seguir treballant amb la variable "preuTotal"

</script>
```

Para que la función devuelva un valor, solo hace falta escribir la palabra reservada **return** junto al nombre de la variable que se quiere devolver. En el ejemplo anterior, la ejecución de la función llega a la instrucción **return** precioTotal, y en este momento, devuelve el valor que contiene la variable precioTotal.

Como la función devuelve un valor, en el punto en el que se realiza la llamada, hay que indicar el nombre de una variable en la que se guardara el valor retornado.

Si no se indica ninguna variable de recogida, JavaScript no producirá ningún error pero el valor retornado por la función se perderá.

En este caso tampoco es necesario que la variable que devuelve la función tenga el mismo nombre que la variable que la recoge.

Si la función llega a la instrucción del tipo return, se devuelve el valor y finaliza la ejecución de la función, lo que significa que todas las ordenes posteriores serán ignoradas.

Variables predefinidas

JavaScript posee una serie de funciones predefinidas.

Las más importantes son:

- **isNaN()** -> Determina si el valor introducido como argumento es un número o no, devolviendo true o false.
- **parseInt()** -> Convierte a numérico una cadena.
- **parseFloat()** -> Convierte a decimal una cadena.

Array

Objeto Array:

El objeto **Array** permite crear matrices para guardar valores de diferentes tipos en una sola variable.

Consta de las siguientes propiedades:

Consta de las siguientes propiedades:

| Propiedad | Descripción |
|-------------|---|
| constructor | Devuelve la función que ha creado el objeto |
| prototype | Permite añadir propiedades y métodos al objeto creado |
| length | Devuelve la longitud del objeto |

Consta a su vez de los siguientes métodos:

| Método | Descripción |
|---------------|---|
| concat() | Une dos o más matrices y devuelve el resultado |
| indexOf() | Busca en la matriz un elemento y devuelve su posición |
| join() | Une todos los elementos de la matriz y devuelve una cadena |
| lastIndexOf() | Busca en la matriz el último elemento y devuelve su posición |
| pop() | Borra el último elemento de la matriz y lo devuelve |
| push() | Añade un elemento al final de la matriz y devuelve su longitud |
| reverse() | Invierte el orden de los elementos de una matriz |
| shift() | Borra el primer elemento de la matriz y lo devuelve |
| slice() | Selecciona una parte de la matriz y la devuelve como otra matriz |
| sort() | Ordena los elementos de la matriz |
| splice() | Añade/borra elementos de la matriz |
| toString() | Convierte una matriz en una cadena |
| unshift() | Añade elementos al principio de la matriz y devuelve su resultado |
| valueOf() | Devuelve el valor original del objeto |

Objetos – DOM

Introducción:

Un objeto es un grupo de variables agrupadas, estas variables se denominan propiedades. Los objetos también disponen de una serie de métodos o funciones que realizan operaciones con las variables propias de estos.

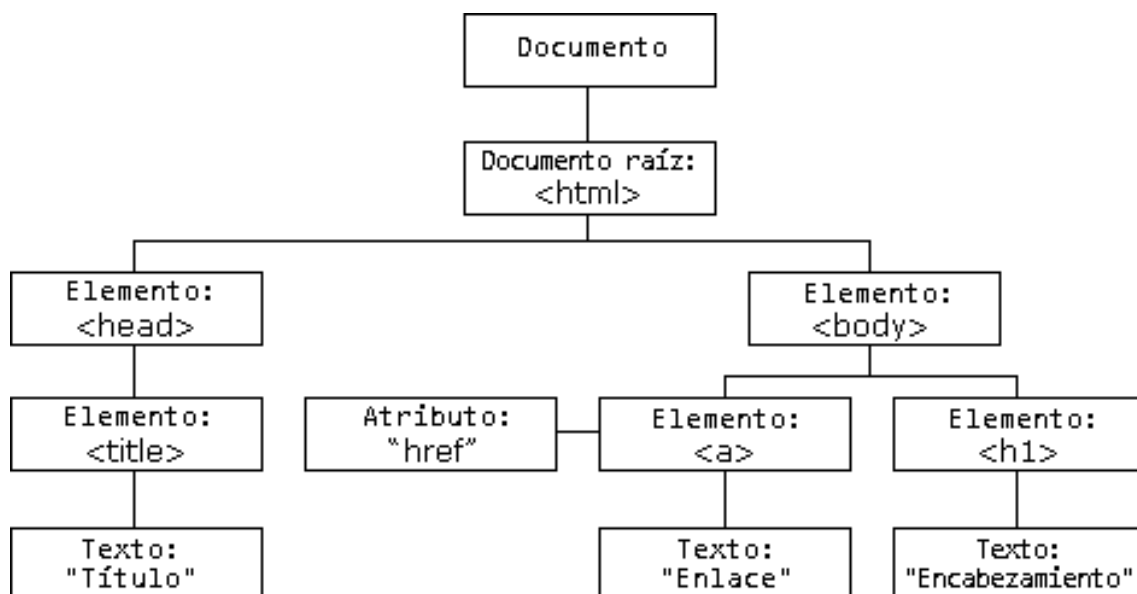
Objetos incorporados:

JavaScript utiliza todo un conjunto de objetos que representan el navegador y sus elementos, el documento HTML, las imágenes, los enlaces, las tablas etc.

Cada uno de estos objetos está definido por sus propiedades, los datos que almacena en su interior y por sus métodos, es decir, las acciones que podemos hacer sobre ellos.

Para comprender a fondo la programación en JavaScript, es imprescindible conocer la jerarquía de objetos que lo forman. Cada uno de estos objetos puede contener varios tipos de elementos, otros objetos JavaScript, propiedades y métodos.

La jerarquía de los objetos JavaScript es la siguiente:



El modelo **DOM** (*Document Object Model*) es una interfaz de programación (API) para documentos HTML y XML válidos, que permite acceder de manera estándar a todos sus elementos como si fueran objetos, con una estructura lógica.

El modelo DOM es un estándar creado por la W3C.

Con el modelo DOM HTML se puede modificar, añadir o eliminar cualquier elemento del documento HTML.

Éste modelo es accesible desde JavaScript.

Ejemplo de cambio de texto en un elemento:

```
var texto = document.getElementById("titulo");
```

```
texto.innerHTML = "Nuevo título";
```

Acceso:

En el modelo DOM HTML el objeto **document** es la página web, es el *padre* de todos los elementos del documento, y el acceso a cualquier elemento del documento se realiza a través de él.

Métodos para encontrar elementos:

| Método | Descripción |
|---|--|
| <code>document.getElementById()</code> | Busca un elemento por su atributo id |
| <code>document.getElementsByTagName()</code> | Busca elementos por su etiqueta HTML |
| <code>document.getElementsByClassName()</code> | Busca elementos por su nombre de clase |
| <code>document.forms[]</code> , <code>document.images[]</code> , <code>document.links[]</code> , <code>document.anchors[]</code> | Busca elementos por los elementos de colección del objeto HTML |

Ejemplos de búsqueda de elementos:

```
// Buscar un elemento por la id="contenedor"

var x = document.getElementById("contenedor");

// Buscar los párrafos que hay en el elemento con id="contenedor"

var parrafos = x.getElementsByTagName("p");

parrafos[0].innerHTML = "Le cambiamos el texto al primer párrafo";

// Buscar elementos span en el documento

var s = document.getElementsByTagName("span");

// Buscar los elementos por el nombre de la clase "encabezamientos"

var e = document.getElementsByClassName("encabezamientos");
```

Desde el modelo DOM HTML se puede acceder a colecciones de objetos que dependen de otros:

- Anclas: **document.anchors.**
- Formularios: **document.forms.**
- Imágenes: **document.images.**
- Enlaces: **document.links**

Ejemplos:

```
// Imprime el valor de todos los elementos input del primer formulario

for (var i = 0; i < document.forms[0].length; i++) {

    document.writeln(document.forms[0].elements[i].value + "<br>");

}

// Cambiar el atributo src de la primera imagen del documento

document.images[0].src = "nueva/imagen.png";
```

Modificación contenido:

Para modificar el contenido, atributos o estilos de text CSS se hace uso de las siguientes propiedades y metodos:

| Método/propiedad | Descripción |
|---|---|
| <code>elemento.innerHTML="contenido"</code> | Cambia el contenido de un elemento HTML |
| <code>elemento.attribute="valor"</code> | Cambia el valor del <i>atributo</i> de un elemento HTML |
| <code>elemento.setAttribute(atributo, valor)</code> | Añade un <i>atributo</i> a un elemento HTML y le asigna un <i>valor</i> |
| <code>elemento.style.propiedad="estilo"</code> | Cambiar el estilo CSS <i>propiedad</i> de un elemento HTML |

// Cambiar el texto del elemento identificado como 'id_elemento'

```
var x = document.getElementById("id_elemento");
```

```
x.innerHTML = "Nuevo texto al elemento";
```

```
x.style.color = "rgb(0, 0, 255)"; // color de texto azul
```

```
x.className = "claseCSS"; // Cambiarle el valor del atributo class
```

// Asignar una clase CSS a un elemento de formulario

```
var i = document.getElementById("id_input");
```

```
i.size = "60"; // Le cambia el valor del tamaño del elemento input
```

```
i.setAttribute("class", "claseCSS"); // Le añade el atributo class
```

```
i.className = i.className + " otraClaseCSS"; Añadirle otra clase
```

Añadir, eliminar o modificar elementos HTML:

Se pueden añadir, eliminar o modificar elementos HTML con los siguientes métodos:

| Método | Descripción |
|--|---|
| <code>document.createElement(<i>elemento</i>)</code> | Crea un nuevo elemento HTML <i>elemento</i> . |
| <code>document.removeChild(<i>nodo</i>)</code> | Borra un nodo hijo (un elemento HTML). |
| <code>document.appendChild(<i>nodo</i>)</code> | Añade un nodo (un elemento HTML) |
| <code>document.replaceChild(<i>nodo</i>)</code> | Reemplaza un nodo (un elemento HTML) |
| <code>document.createTextNode(<i>texto</i>)</code> | Crear nodo de texto para un elemento HTML |

// Crear una nueva lista

var u = document.createElement("UL");

// Crear un nuevo elemento de lista

var l = document.createElement("LI");

// Crear un nodo de texto

var t = document.createTextNode("Elemento lista");

// Añadir el nodo de texto al elemento de lista

l.appendChild(t);

// Añadir el elemento de lista a la nueva lista

u.appendChild(l);

// Añadir la nueva lista al documento

document.body.appendChild(l);

Window

Introducción:

El modelo BOM (**B**rowser **O**bject **M**odel) o **M**odelo de **O**bjeto de **N**avegador, permite acceder a algunas características del navegador a través del JavaScript.

A diferencia del DOM que es un estándar creado por la W3C, el BOM no es un estándar oficial.

Los navegadores modernos han implementado (al menos) los mismos métodos y propiedades para los siguientes objetos:

Window (ventana), **Screen** (pantalla), **Location** (localización), **History** (historial), **Navigator** (navegador), **PopAlert** (cajas de alerta), **Timing** (temporización).

Objeto Window:

El objeto **window** está soportado por la mayoría de los navegadores, y representa la ventana del navegador.

Todos los objetos globales del JavaScript, funciones y variables automáticamente se convierten en elementos del objeto **window**.

- Las variables globales son propiedades el objeto **window**.
- Las funciones globales son métodos del objeto **window**.
- Cada objeto del documento (del DOM HTML) es una propiedad del objeto **window**

```
window.document.getElementById("id_elemento");
```

Es igual a:

```
document.getElementById("id_elemento");
```

Método Open():

El objeto **window** tiene seis métodos, el primero de ellos es el método **window.open()**, que sirve para abrir una ventana.

Sintaxis:

[referencia =] window.open([URL], [nombre], [opciones], [reemplazar]);

- **URL**: opcional, dirección del documento, si no se especifica abre con *about:blank*.
- **nombre**: opcional, las mismas opciones que el atributo *target* de los enlaces HTML.
 - **_blank**: abre una ventana nueva (por defecto).
 - **_parent**: abre en el marco padre.
 - **_self**: en la ventana actual.
 - **_top**: reemplaza cualquier conjunto de marcos que exista.
 - **nombre**: nombre de la ventana.
- **opciones**: opcional, opciones relacionadas con la ventana:
 - **channelmode**: ventana en modo *teatro*. Por defecto es no. Sólo IE.
 - **fullscreen**: ventana en modo pantalla completa. Por defecto es no. Debe de estar en modo *teatro*. Sólo IE.
 - **height**: altura en pixels, mínimo 100.
 - **left**: desplazamiento horizontal en pixels, sólo valores positivos.
 - **location**: activar o no la barra de navegación. Sólo Opera.
 - **menubar**: activar o no la barra de menús.
 - **resizable**: ventana redimensionable o no. Sólo IE.
 - **scrollbars**: activar o no la barras de desplazamiento. Sólo IE, Firefox y Opera.
 - **status**: activar o no la barra de estado.
 - **titlebar**: activar o no la barra de título.
 - **top**: desplazamiento vertical en pixels, sólo valores positivos.
 - **width**: anchura de la ventana en pixels, mínimo 100.

- **reemplazar:** opcional, si crea una nueva entrada o reemplaza la actual en el historial de navegación:
 - **true:** reemplaza la entrada actual del documento.
 - **false:** crea una nueva entrada.
- **referencia:** referencia a la ventana abierta para enviar contenido

Ejemplos:

```
window.open("pag1.html", "vent1", "menubar=yes,toolbar=no,location=no");  
  
var v2 = window.open("", "vent2", "menubar=1,toolbar=0,location=0");  
  
v2.document.writeln("<p>Párrafo en la ventana 2.</p>");
```

Método close():

El método **window.close()** cierra la ventana. Se usa en conjunción con el método **window.open()**.

Sintaxis: `window.close()` | `referencia.close()`

Ejemplos:

```
var vent1 = window.open("", "pag1", "width=640,height=480");  
  
vent1.document.writeln("<p>¡Hola Mundo!</p>");  
  
function cerrar()  
{  
  
    vent1.close()  
  
}
```

Método moveTo():

El método **window.moveTo()** desplaza la ventana en posiciones absolutas con origen de coordenadas la parte superior izquierda de la pantalla.

Sintaxis: `window.moveTo(x, y);`

- **x**: obligatorio, coordenada horizontal a mover la ventana, admite valores negativos.
- **y**: obligatorio, coordenada vertical a mover la ventana, admite valores negativos.

Ejemplo:

```
var miVentana = window.open("", "myWin", "width=720,height=576");  
miVentana.moveTo(100, 200); // Posición absoluta (100, 200)
```

Método moveBy():

El método **window.moveBy()** hace el desplazamiento de la ventana desde su posición actual.

Sintaxis: `window.moveBy(dx, dy);`

- **dx**: obligatorio, desplazamiento horizontal a mover la ventana, admite valores negativos.
- **dy**: obligatorio, desplazamiento vertical a mover la ventana, admite valores negativos.

Ejemplo:

```
var miVentana = window.open("", "myWin", "width=720,height=576");  
miVentana.moveTo(100, 200); // Posición absoluta (100, 200)  
miVentana.moveBy(-30,-60); // Posición absoluta (70, 140)
```


Método `resizeTo()`:

El método **`window.resizeTo()`** ajusta la ventana a unas dimensiones dadas.

Sintaxis: `window.resizeTo(w, h);`

Ejemplo:

```
var miVentana = window.open("", "ventana", "width=512,height=384");  
miVentana.document.writeln("<p>Ventana de 512x384 pixels</p>");  
miVentana.resizeTo(640, 480);  
miVentana.document.writeln("<p>Ventana de 640x480 pixels</p>");
```

Método `resizeBy()`:

El método **`window.resizeBy()`** redimensiona una ventana a unos valores relativos.

Sintaxis: `window.resizeBy(dw, dh);`

Ejemplo:

```
var miVentana = window.open("", "ventana", "width=512,height=384");  
miVentana.document.writeln("<p>Ventana de 512x384 pixels</p>");  
miVentana.resizeTo(640, 480);  
miVentana.document.writeln("<p>Ventana de 640x480 pixels</p>");  
miVentana.resizeBy(160, 120);  
miVentana.document.writeln("<p>Ventana de 800x600 pixels</p>");
```

Objeto Location:

El objeto window.location tiene información sobre la URL de la página actual, se puede escribir sin el window y tiene las siguiente propiedades:

| Propiedad | Descripción |
|-----------|--|
| hash | Devuelve o asigna la porción del ancla de la URL |
| host | Devuelve o asigna el nombre del host y el número de puerto de la URL |
| hostname | Devuelve o asigna el nombre del host de la URL |
| href | Devuelve o asigna la URL completa |
| origin | Devuelve el protocolo, el nombre de host y el número de puerto de una URL |
| pathname | Devuelve o asigna el camino completo de una URL |
| port | Devuelve o asigna el número de puerto de una URL |
| protocol | Devuelve o asigna el protocolo de la URL |
| search | Devuelve o asigna la porción de los parámetros de consulta de la URL (GET) |

Dispone de los siguientes métodos:

| Método | Descripción |
|-----------|---|
| assign() | Carga un nuevo documento |
| reload() | Recarga el documento actual |
| replace() | Reemplaza el documento actual por uno nuevo |

Ejemplos:

<script>

document.write(window.location.href); // Escribir la URL del documento

**document.writeln("
");**

document.write(location.pathname); // Escribir el camino

**document.writeln("
");**

document.write(location.protocol); // Escribir el protocolo

**document.writeln("
");**

location.reload(); // Recargar la página

location.assign("http://www.fundaciocim.org/");

location.replace("http://www.google.es/");

</script>

Objeto History:

El objeto window.history contiene el historial de navegacion del navegador. Como el objeto location, se puede escribir sin window y contiene la siguiente propiedad:

| Propiedad | Descripción |
|-----------|--|
| length | Devuelve el número de URLs del historial |

Y los siguientes métodos:

| Método | Descripción |
|-----------|--|
| back() | Carga la URL anterior del historial |
| forward() | Carga la siguiente URL del historial |
| go() | Carga una URL específica del historial |

Ejemplos:

```
<script>
```

```
document.write("Número total de URLs: " + history.length);
```

```
document.writeln("<br>");
```

```
history.back(); // Equivale a pulsar el botón retroceder
```

```
history.forward(); // Equivale a pulsar el botón de avanzar
```

```
history.go(-3); // Ir a la URL de tres posiciones anteriores.
```

```
</script>
```

Objeto Navigator:

El objeto window.navigator contiene información acerca del propio navegador. Se puede usar sin escribir window y tiene las siguientes propiedades:

| Propiedad | Descripción |
|---------------|---|
| appName | Devuelve el código de nombre del navegador |
| appVersion | Devuelve el nombre del navegador |
| cookieEnabled | Devuelve la información de versión del navegador |
| language | Indica si están activas las cookies del navegador |
| onLine | Devuelve el idioma predefinido del navegador |
| platform | Indica si el navegador está en línea |
| product | Devuelve información acerca de a qué plataforma está compilado |
| userAgent | Devuelve el nombre del motor del navegador |
| | Devuelve la cabecera user-agent que envía el navegador al servidor para identificarse |

Y el siguiente método:

| Método | Descripción |
|---------------|--|
| javaEnabled() | Indica si está o no activa la máquina virtual de |

Java

Ejemplos:

<script>

```
document.writeln("Código: " + navigator.appCodeName + "<br>");  
document.writeln("Nombre: " + navigator.appName + "<br>");  
document.writeln("Versión: " + navigator.appVersion + "<br>");  
document.writeln("¿Cookies?: " + navigator.cookieEnabled + "<br>");  
document.writeln("Idioma: " + navigator.language + "<br>");  
document.writeln("¿En línea?: " + navigator.onLine + "<br>");  
document.writeln("Plataforma: " + navigator.platform + "<br>");  
document.writeln("Producto: " + navigator.product + "<br>");  
document.writeln("Agente: " + navigator.userAgent + "<br>");  
document.writeln("¿Java?: " + navigator.javaEnabled() + "<br>");
```

</script>

Acceso a elementos de la ventana padre:

Para acceder a los elementos de la ventana padre mediante el objeto documento se utiliza la propiedad **opener**. Su sintaxis sería la siguiente:

Ejemplos:

```
window.opener.document.getElementById('Title').value;
```

Eventos

Introducción:

Un evento es un hecho o suceso que ocurre durante la ejecución de un programa.

Desde el lenguaje JavaScript se pueden administrar una serie de eventos:

- Eventos de ratón: por ejemplo la pulsación de los botones del ratón o mover un elemento de la página.
- Eventos de página: cuando se carga una página.
- Eventos con imágenes: cuando se cargan imágenes.
- Eventos de formularios: cuando se selecciona un campo de entrada, cambia su contenido o se envía un formulario.
- Eventos de teclado: cuando se pulsa una tecla.

Añadir eventos a elementos HTML

A los elementos HTML se les puede asociar uno o varios eventos para que ejecuten un código en JavaScript.

Ejemplos de asignación de administradores de eventos a elementos HTML:

```
<h1 onclick="this.innerHTML='¡Oops!'" onmouseover="this.style.color='red'">
```

¡Clica sobre el texto!

```
</h1>
```

Lista de eventos:

| Evento | Descripción | Elementos para los que está definido |
|------------|--|--|
| onblur | Deseleccionar el elemento | <button>, <input>, <label>, <select>, <textarea>, <body> |
| onchange | Deseleccionar un elemento que se ha modificado | <input>, <select>, <textarea> |
| onclick | Pinchar y soltar el ratón | Todos los elementos |
| ondblclick | Pinchar dos veces seguidas con el ratón | Todos los elementos |
| onfocus | Seleccionar un elemento | <button>, <input>, <label>, <select>, <textarea>, <body> |
| onkeydown | Pulsar una tecla (sin soltar) | Elementos de formulario y <body> |
| onkeypress | Pulsar una tecla | Elementos de formulario y <body> |

| Evento | Descripción | Elementos para los que está definido |
|-------------|---|---|
| onkeyup | Soltar una tecla pulsada | Elementos de formulario y <code><body></code> |
| onload | La página se ha cargado completamente | <code><body></code> |
| onmousedown | Pulsar (sin soltar) un botón del ratón | Todos los elementos |
| onmousemove | Mover el ratón | Todos los elementos |
| onmouseout | El ratón "sale" del elemento (pasa por encima de otro elemento) | Todos los elementos |
| onmouseover | El ratón "entra" en el elemento (pasa por encima del elemento) | Todos los elementos |
| onmouseup | Soltar el botón que estaba pulsado en el | Todos los elementos |

| Evento | Descripción | Elementos para los que está definido |
|----------|--|--------------------------------------|
| | ratón | |
| onreset | Inicializar el formulario (borrar todos sus datos) | <form> |
| onresize | Se ha modificado el tamaño de la ventana del navegador | <body> |
| onselect | Seleccionar un texto | <input>, <textarea> |
| onsubmit | Enviar el formulario | <form> |
| onunload | Se abandona la página (por ejemplo al cerrar el navegador) | <body> |

Los eventos más utilizados en las aplicaciones web tradicionales son **onload** para esperar a que se cargue la página por completo, los eventos **onclick**, **onmouseover**, **onmouseout** para controlar el ratón y **onsubmit** para controlar el envío de los formularios.

Manejadores de eventos:

Un evento de JavaScript por sí mismo carece de utilidad. Para que los eventos resulten útiles, se deben asociar funciones o código JavaScript a cada evento. De esta forma, cuando se produce un evento se ejecuta el código indicado, por lo que la aplicación puede responder ante cualquier evento que se produzca durante su ejecución.

Las funciones o código JavaScript que se definen para cada evento se denominan "manejador de eventos" y como JavaScript es un lenguaje muy flexible, existen varias formas diferentes de indicar los manejadores:

Manejadores como atributos de los elementos XHTML.

Manejadores como funciones JavaScript externas.

Manejadores "semánticos".

Manejadores de eventos como atributos XHTML

Se trata del método más sencillo y a la vez menos profesional de indicar el código JavaScript que se debe ejecutar cuando se produzca un evento. En este caso, el código se incluye en un atributo del propio elemento XHTML. En el siguiente ejemplo, se quiere mostrar un mensaje cuando el usuario pinche con el ratón sobre un botón:

```
<input type="button" value="Pinchame y verás" onclick="alert('Gracias por pinchar');" />
```

En este método, se definen atributos XHTML con el mismo nombre que los eventos que se quieren manejar. El ejemplo anterior sólo quiere controlar el evento de pinchar con el ratón, cuyo nombre es onclick. Así, el elemento XHTML para el que se quiere definir este evento, debe incluir un atributo llamado onclick.

El contenido del atributo es una cadena de texto que contiene todas las instrucciones JavaScript que se ejecutan cuando se produce el evento. En este caso, el código JavaScript es muy sencillo (**alert('Gracias por pinchar');**), ya que solamente se trata de mostrar un mensaje.

En este otro ejemplo, cuando el usuario pincha sobre el elemento <div> se muestra un mensaje y cuando el usuario pasa el ratón por encima del elemento, se muestra otro mensaje:

```
<div onclick="alert('Has pinchado con el ratón');" onmouseover="alert('Acabas de pasar el ratón por encima');">
```

Puedes pinchar sobre este elemento o simplemente pasar el ratón por encima

```
</div>
```

Este otro ejemplo incluye una de las instrucciones más utilizadas en las aplicaciones JavaScript más antiguas:

```
<body onload="alert('La página se ha cargado completamente');">
```

...

```
</body>
```

El mensaje anterior se muestra después de que la página se haya cargado completamente, es decir, después de que se haya descargado su código HTML, sus imágenes y cualquier otro objeto incluido en la página.

El evento onload es uno de los más utilizados ya que las funciones que permiten acceder y manipular los nodos del árbol DOM solamente están disponibles cuando la página se ha cargado completamente.

Manejadores de eventos y variable this

JavaScript define una variable especial llamada `this` que se crea automáticamente y que se emplea en algunas técnicas avanzadas de programación. En los eventos, se puede utilizar la variable `this` para referirse al elemento XHTML que ha provocado el evento. Esta variable es muy útil para ejemplos como el siguiente:

Cuando el usuario pasa el ratón por encima del `<div>`, el color del borde se muestra de color negro. Cuando el ratón sale del `<div>`, se vuelve a mostrar el borde con el color gris claro original.

Elemento `<div>` original:

```
<div id="contenidos" style="width:150px; height:60px; border:thin solid silver">
```

Sección de contenidos...

```
</div>
```

Si no se utiliza la variable `this`, el código necesario para modificar el color de los bordes, sería el siguiente:

```
<div id="contenidos" style="width:150px; height:60px; border:thin solid silver"  
onmouseover="document.getElementById('contenidos').style.borderColor='black';"  
onmouseout="document.getElementById('contenidos').style.borderColor='silver';">
```

Sección de contenidos...

```
</div>
```

El código anterior es demasiado largo y demasiado propenso a cometer errores. Dentro del código de un evento, JavaScript crea automáticamente la variable `this`, que hace referencia al elemento XHTML que ha provocado el evento. Así, el ejemplo anterior se puede reescribir de la siguiente manera:

```
<div id="contenidos" style="width:150px; height:60px; border:thin solid silver"
onmouseover="this.style.borderColor='black';"
onmouseout="this.style.borderColor='silver';">
```

Sección de contenidos...

```
</div>
```

El código anterior es mucho más compacto, más fácil de leer y de escribir y sigue funcionando correctamente aunque se modifique el valor del atributo id del <div>.

Manejadores de eventos como funciones externas

La definición de los manejadores de eventos en los atributos XHTML es el método más sencillo pero menos aconsejable de tratar con los eventos en JavaScript. El principal inconveniente es que se complica en exceso en cuanto se añaden algunas pocas instrucciones, por lo que solamente es recomendable para los casos más sencillos.

Si se realizan aplicaciones complejas, como por ejemplo la validación de un formulario, es aconsejable agrupar todo el código JavaScript en una función externa y llamar a esta función desde el elemento XHTML.

Siguiendo con el ejemplo anterior que muestra un mensaje al pinchar sobre un botón:

```
<input type="button" value="Pinchame y verás" onclick="alert('Gracias por pinchar');" />
```

Utilizando funciones externas se puede transformar en:

```
function muestraMensaje()
```

```
{
```

```
    alert('Gracias por pinchar');
```

```
}
```

```
<input type="button" value="Pinchame y verás" onclick="muestraMensaje()" />
```

Esta técnica consiste en extraer todas las instrucciones de JavaScript y agruparlas en una función externa. Una vez definida la función, en el atributo del elemento XHTML se incluye el nombre de la función, para indicar que es la función que se ejecuta cuando se produce el evento.

La llamada a la función se realiza de la forma habitual, indicando su nombre seguido de los paréntesis y de forma opcional, incluyendo todos los argumentos y parámetros que se necesiten.

El principal inconveniente de este método es que en las funciones externas no se puede seguir utilizando la variable `this` y por tanto, es necesario pasar esta variable como parámetro a la función:

```
function resalta(elemento)
{
    switch(elemento.style.borderColor)
    {
        case 'silver':
        case 'silver silver silver silver':
        case '#c0c0c0':
            elemento.style.borderColor = 'black';
            break;
        case 'black':
        case 'black black black black':
        case '#000000':
            elemento.style.borderColor = 'silver';
            break;
    }
}
```

```
<div style="width:150px; height:60px; border:thin solid silver" onmouseover="resalta(this)"  
onmouseout="resalta(this)">
```

Sección de contenidos...

```
</div>
```

En el ejemplo anterior, la función externa es llamada con el parámetro this, que dentro de la función se denomina elemento. La complejidad del ejemplo se produce sobre todo por la forma en la que los distintos navegadores almacenan el valor de la propiedad borderColor.

Mientras que Firefox almacena (en caso de que los cuatro bordes coincidan en color) el valor black, Internet Explorer lo almacena como black black black black y Opera almacena su representación hexadecimal #000000.

Cookies

Introducción:

La **cookies** (*galletas*) son pequeños archivos de texto con datos almacenados en el ordenador.

Dado el funcionamiento cliente-servidor de Internet, donde los servidores una vez terminan de enviar los datos cierran la conexión, el servidor *olvida* todo acerca del usuario.

Para resolver éste problema se crearon las **cookies**, que permiten *recordar* la información del usuario cuando, por ejemplo:

- Visita una web, su nombre se almacena en una cookie.
- Vuelve a la misma web, la cookie le *recuerda* el nombre al servidor.

Se almacenan como pares nombre=valor:

Usuario = Juan Nadie

Lectura, creación, modificación y borrado de cookies:

Para crear, leer, modificar y borrar cookies en JavaScript, se hace uso de la propiedad **document.cookie**.

Para crear una cookie: **document.cookie = "usuario=Juan Nadie";**

Por defecto se borran cuando se cierra la conexión, pero se puede añadir información de caducidad:

document.cookie = "usuario=Juan Nadie; expires=Fri, 28 Mar 2014 12:30:00 GMT";

// Caduca el viernes 28 de marzo de 2014 a las 12:30

Por defecto pertenece a la página actual, pero con el parámetro *path* se le puede indicar a qué página pertenece:

document.cookie = "usuario=Juan Nadie; expires=Fri, 28 Mar 2014 12:30:00 GMT; path=/";

// Pertenece a la raíz de la web

Para leer una cookie basta con asignarla a una variable:

```
var x = document.cookie;
```

Se ha de tener en cuenta que devuelve todas las cookies en una sola cadena de texto, en pares nombre_cookie=valor_cookie.

Para modificar una cookie, de la misma forma que se crea:

```
document.cookie = "usuario=John Smith; expires=Fri, 28 Mar 2014 13:30:00 GMT; path=/";
```

Para borrar una cookie, basta con poner una fecha de caducidad antigua:

```
document.cookie = "usuario=; expires=Thu, 01 Jan 1970 00:00:00 GMT;";
```

La propiedad **document.cookie** puede parecer una cadena de caracteres normal, pero no es así.

Aunque se escriba una gran cadena de cookie en document.cookie, al leerla de nuevo sólo se podrá ver su par nombre=valor.

Si se crea una nueva cookie, las existentes no se borran ni sobrescriben, se añade.

Si se desea encontrar una cookie concreta, se tendrá que utilizar las funciones de JavaScript para buscar cadenas:

// Ejemplo de obtención de una cookie, retorna la posición de inicio

```
var miCookie = document.cookie.indexOf("miCookie");
```

String

STRING:

El objeto **String** se emplea para manipular cadenas de caracteres principalmente.

Existen varias formas de declarar cadenas:

```
var cadena1 = new String("Creando una cadena");
```

```
var cadena2 = "Creando otra cadena";
```

Dispone de las siguientes propiedades:

| Propiedad | Descripción |
|-------------|---|
| constructor | Devuelve la función que ha creado el objeto |
| prototype | Permite añadir propiedades y métodos al objeto creado |
| length | Devuelve la longitud de la cadena |

Dispone de los siguientes métodos:

| Método | Descripción |
|---------------------------------|---|
| <code>charAt(indice)</code> | Devuelve el caracter especificado por <i>indice</i> |
| <code>charCodeAt(indice)</code> | Devuelve el valor Unicode del carácter especificado por <i>indice</i> |
| <code>concat()</code> | Une dos o más cadenas, y devuelve el resultado |
| <code>fromCharCode()</code> | Convierte valores Unicode a caracteres |
| <code>indexOf()</code> | Devuelve la primera posición de la cadena especificada |
| <code>lastIndexOf()</code> | Devuelve la última posición de la cadena especificada |
| <code>localeCompare()</code> | Compara dos cadenas en la localización actual |
| <code>match()</code> | Busca coincidencias entre una expresión regular y una cadena |
| <code>replace()</code> | Reemplaza una cadena (o expresión regular) buscada por otra |
| <code>search()</code> | Busca coincidencias entre una expresión regular y una cadena |
| <code>slice()</code> | Extrae una parte de una cadena y devuelve el resultado |
| <code>split()</code> | Divide una cadena en una matriz de cadenas |

| | |
|----------------------------------|---|
| <code>substr()</code> | Extrae caracteres de una cadena, desde un inicio [y longitud] |
| <code>substring()</code> | Extrae caracteres entre dos índices especificados |
| <code>toLocaleLowerCase()</code> | Convierte a minúsculas, según la configuración local del host |
| <code>toLocaleUpperCase()</code> | Convierte a mayúsculas, según la configuración local del host |
| <code>toLowerCase()</code> | Convierte a minúsculas |
| <code>toString()</code> | Devuelve el valor de un objeto String |
| <code>toUpperCase()</code> | Convierte a mayúsculas |
| <code>trim()</code> | Elimina espacios sobrantes delante y detrás de la cadena |
| <code>valueOf()</code> | Devuelve el valor original de un objeto String |

Ejemplos:

```
var cadena1 = "¡Hola Mundo!";
```

```
var matrizCadenas = cadena1.split(' ');
```

```
var cadena2 = cadena1.replace("Hola", "Hello");
```

```
var posicion = cadena1.search("Mundo");
```

```
var cadena3 = "   Hello World!   ";
```

```
var cadena4 = cadena3.trim(); // Devuelve "Hellow World!"
```

Number

NUMBER:

El objeto **Number** se usa para almacenar valores numéricos simples.

La asignación de un número a una variable se puede hacer de dos formas diferentes:

```
var n = new Number(100);
```

```
var m = 300;
```

```
var x = new Number(1.122334);
```

```
var y = 2.132;
```

Consta de las siguientes propiedades:

| Propiedad | Descripción |
|-------------------|---|
| constructor | Devuelve la función que ha creado el prototipo de objeto Number |
| MAX_VALUE | Devuelve el mayor número que puede manipular el JavaScript |
| MIN_VALUE | Devuelve el menor número que puede manipular el JavaScript |
| NEGATIVE_INFINITY | Representa el infinito negativo |
| NaN | Representa el valor "Not-a-Number" (no es un número) |
| POSITIVE_INFINITY | Representa el infinito positivo |
| prototype | Permite añadir propiedades y métodos a |

un objeto

Consta de los siguientes métodos:

| Método | Descripción |
|------------------|---|
| toExponential(x) | Convierte un número a su notación exponencial |
| toFixed(x) | Formatea un número con un número x de decimales. |
| toPrecision(x) | Formatea un número a x dígitos |
| toString() | Convierte un objeto Number a cadena de caracteres |
| valueOf() | Devuelve el valor original de un objeto Number |

Ejemplos:

```
document.writeln(n.toFixed(5)); // Escribe con 5 decimales
```

```
var s = m.toString(); // Convierte el valor de m en cadena
```

Math

MATH:

El objeto **Math** permite realizar operaciones aritméticas.

Incluye una serie de métodos y valores constantes:

// Tomar el valor de la constante Pi

var x = Math.PI;

// Calcular la raíz cuadrada de 16

var y = Math.sqrt(16);

// Obtener el seno del ángulo Pi

var z = Math.sin(x)

Consta de las siguientes propiedades:

| Propiedad | Descripción |
|-----------|---|
| E | Devuelve el número de Euler |
| LN2 | Devuelve el logaritmo neperiano de 2 |
| LN10 | Devuelve el logaritmo neperiano de 10 |
| LOG2E | Devuelve el logaritmo en base 2 del número E |
| LOG10E | Devuelve el logaritmo en base 10 del número E |
| PI | Devuelve el número Pi |
| SQRT1_2 | Devuelve la raíz cuadrada de 1 entre 2 |
| SQRT2 | Devuelve la raíz cuadrada de 2 |

Consta de los siguientes métodos:

| Método | Descripción |
|-------------------------------|--|
| <code>abs(x)</code> | Devuelve el valor absoluto de x |
| <code>acos(x)</code> | Devuelve el arcocoseno de x en radianes |
| <code>asin(x)</code> | Devuelve el arcoseno de x en radianes |
| <code>atan(x)</code> | Devuelve el arcotangente de x en radianes |
| <code>atan2(y,x)</code> | Devuelve el arcotangente del cociente de dos números |
| <code>ceil(x)</code> | Redondea al número entero más próximo por exceso |
| <code>cos(x)</code> | Devuelve el coseno de x en radianes |
| <code>exp(x)</code> | Devuelve el número E elevado a x |
| <code>log(x)</code> | Devuelve el logaritmo neperiano de x |
| <code>max(x,y,z,...,n)</code> | Devuelve el valor máximo de un conjunto de valores |
| <code>min(x,y,z,...,n)</code> | Devuelve el valor mínimo de un conjunto de valores |
| <code>pow(x,y)</code> | Devuelve el valor de x elevado a y |
| <code>random()</code> | Devuelve un número aleatorio entre 0 y 1 |
| <code>round(x)</code> | Redondea al número entero más próximo |
| <code>sin(x)</code> | Devuelve el seno de x en radianes |
| <code>sqrt(x)</code> | Devuelve la raíz cuadrada de x |

Ejemplos:

```
var e = Math.E; // Copie el valor del número E  
  
document.writeln(Math.round(4.7)); // Escribe 5  
  
document.writeln(Math.random()); // Escribe un número en 0 y 1  
  
document.writeln(Math.floor(Math.random() * 11));  
  
document.writeln((1 - 0.9) * 10); // Escribe 0.9999999999999998
```

Date

Objeto Date:

El objeto **Date** se usa para trabajar con fechas y horas.

Para crear un objeto de fecha hay que usar la palabra clave **new**. Hay cuatro formas:

```
var fecha = new Date(); // Fecha actual
```

```
var fecha = new Date(milisegundos); // Milisegundos desde 01/01/1970
```

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

```
var fecha = new Date(año, mes, día, horas, minutos, segundos, milisegundos);
```

Ejemplos:

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

```
var fecha1 = new Date("October 13, 1975 11:13:00");
```

```
var fecha2 = new Date(75, 10, 13);
```

```
var fecha3 = new Date(75, 10, 13, 12, 30, 0);
```

Consta de las siguientes propiedades:

| Propiedad | Descripción |
|-------------|---|
| constructor | Devuelve la función que ha creado el objeto |
| prototype | Permite añadir propiedades y métodos al objeto creado |

Consta a su vez de los siguientes métodos:

| Método | Descripción |
|---------------------|--|
| getDate() | Devuelve el día del mes (de 1 al 31) |
| getDay() | Devuelve el día de la semana (de 0 al 6) |
| getFullYear() | Devuelve el años completo en cuatro dígitos |
| getHours() | Devuelve la hora en formato 24h |
| getMilliseconds() | Devuelve los milisegundos (de 0 al 999) |
| getMinutes() | Devuelve los minutos |
| getMonth() | Devuelve el mes (del 0 a 11) |
| getSeconds() | Devuelve los segundos |
| getTime() | Devuelve los milisegundos desde el 01/01/1970 a las 00:00:00 |
| getTimezoneOffset() | Devuelve la diferencia en minutos entre la hora UTC y la local |
| getUTCDate() | Devuelve el día del mes en formato UTC (de 1 al 31) |
| getUTCDay() | Devuelve el día de la semana en formato UTC (del 0 al 6) |
| getUTCFullYear | Devuelve el año en cuatro cifras, formato UTC |

| Mètode | Descripció |
|-----------------------------------|--|
| <code>getUTCHours()</code> | Devuelve la hora en formato UTC (24h) |
| <code>getUTCMilliseconds()</code> | Devuelve los milisegundos en formato UTC (del 0 al 999) |
| <code>getUTCMinutes()</code> | Devuelve los minutos en formato UTC |
| <code>getUTCMonth()</code> | Devuelve el mes en formato UTC (del 0 al 11) |
| <code>getUTCSeconds()</code> | Devuelve los segundos en formato UTC |
| <code>parse()</code> | Procesar una fecha en cadena y devuelve el número de milisegundos desde el 1 de enero de 1970 a las 00:00:00 |
| <code>setDate()</code> | Asigna el día del mes a un objeto de fecha |
| <code>setFullYear()</code> | Asigna el año en cuatro cifras a un objeto de fecha |
| <code>setHours()</code> | Asigna la hora a un objeto de fecha |
| <code>setMilliseconds()</code> | Asigna los milisegundos a un objeto de fecha |
| <code>setMinutes()</code> | Asigna los minutos a un objeto de fecha |
| <code>setMonth()</code> | Asigna el mes a un objeto de fecha |
| <code>setSeconds()</code> | Asigna los segundos a un objeto de fecha |

| Mètode | Descripció |
|-----------------------------------|--|
| <code>setTime()</code> | Asigna fecha y hora añadiendo o restando un número especificado de milisegundos a/de 01/01/1970 00:00:00 |
| <code>setUTCDate()</code> | Asigna el día del mes a un objeto de fecha, formato UTC |
| <code>setUTCFullYear()</code> | Asigna el año en cuatro cifras aun objeto fecha, formato UTC |
| <code>setUTCHours()</code> | Asigna las horas a un objeto de fecha, formato UTC |
| <code>setUTCMilliseconds()</code> | Asigna los milisegundos a un objeto de fecha, formato UTC |
| <code>setUTCMinutes()</code> | Asigna los minutos a un objeto de fecha, formato UTC |
| <code>setUTCMonth()</code> | Asigna el mes a un objeto de fecha, formato UTC |
| <code>setUTCSeconds()</code> | Asigna los segundos a un objeto de fecha, formato UTC |
| <code>toDateString()</code> | Convierte la parte de la fecha a una cadena legible |
| <code>toISOString()</code> | Devuelve la fecha como una cadena, formato ISO |
| <code>toJSON()</code> | Devuelve la fecha como una cadena, en formato JSON |
| <code>toLocaleDateString()</code> | Devuelve la parte de fecha en cadena, conversión local |

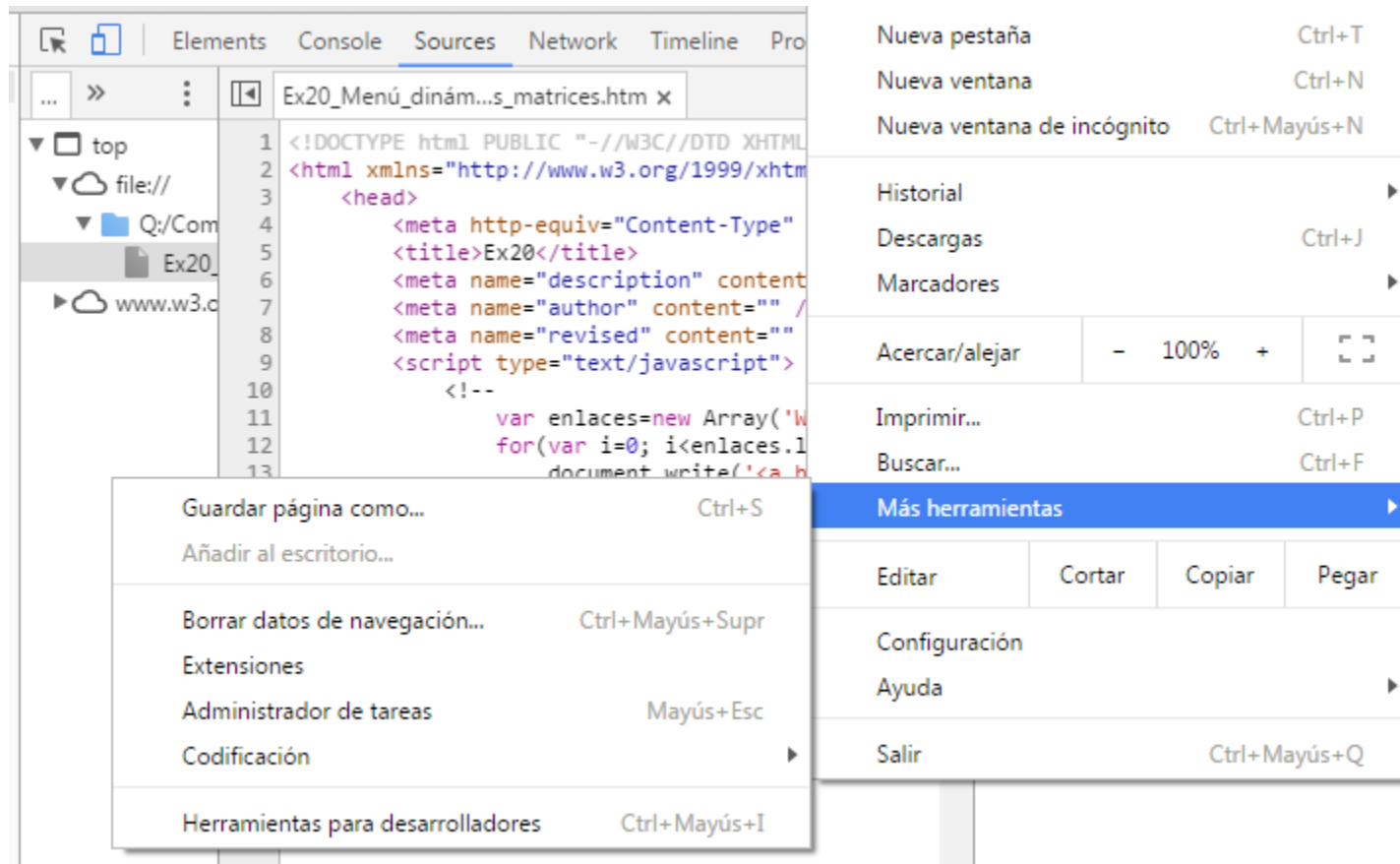
| Mètode | Descripció |
|----------------------|--|
| toLocaleTimeString() | Devuelve la parte de la hora en cadena, conversión local |
| toLocaleString() | Devuelve un objeto fecha en una cadena, conversión local |
| toString() | Devuelve un objeto fecha en una cadena |
| toTimeString() | Devuelve la parte de la hora en formato cadena |
| toUTCString() | Convierte un objeto de fecha en una cadena, formato UTC |
| toISOString() | Devuelve la fecha como una cadena, formato ISO |
| UTC() | Devuelve en milisegundos en formato cadena la fecha desde el 01/01/1970 00:00:00, en formato UTC |
| valueOf() | Devuelve el valor original de un objeto fecha |

Debugging JavaScript

Ejecución paso

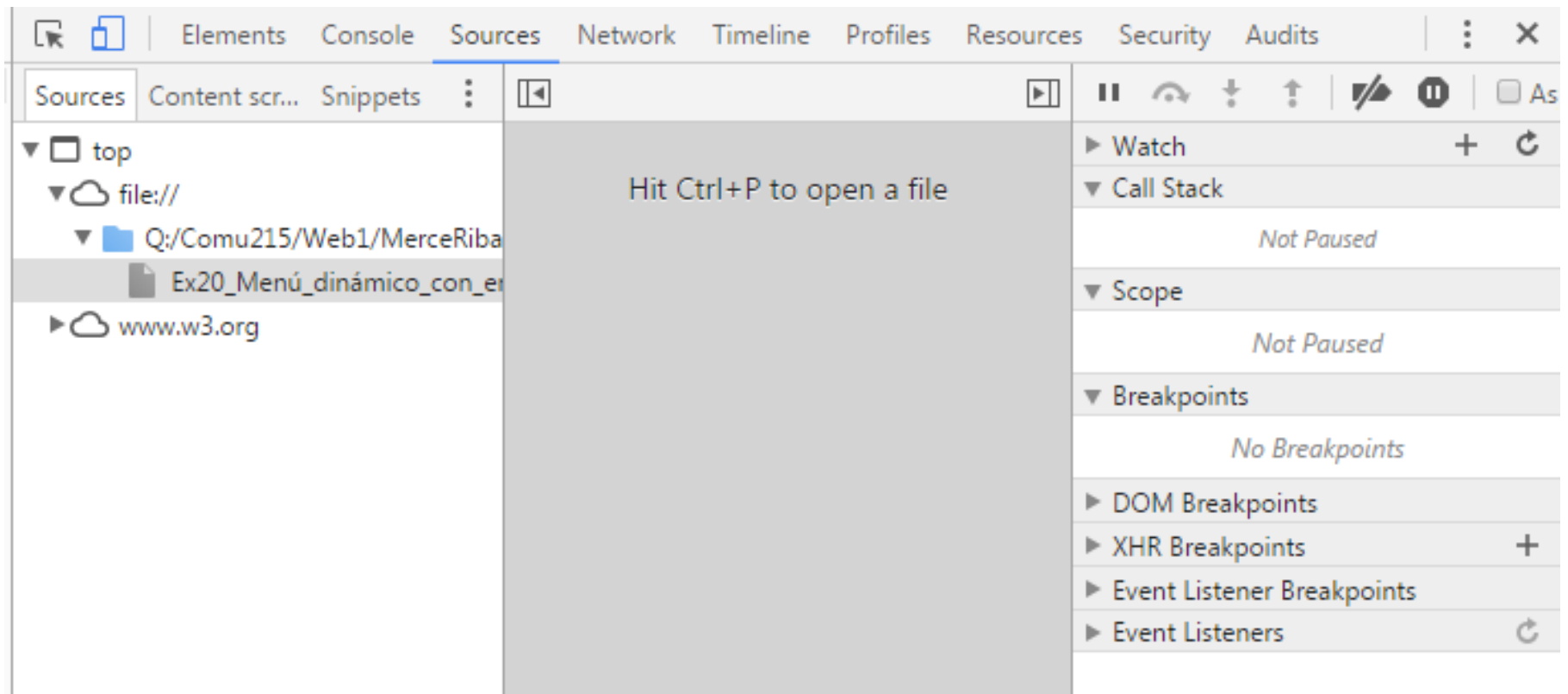
Chrome Herramientas para
desarrolladores

En el navegador chrome tenemos varias maneras de entrar en las herramientas del desarrollador : Tecla rápida de Ctrl + Mayusculas + I
O bien botón superior izquierdo -> Mas herramientas -> Herramientas para desarrolladores

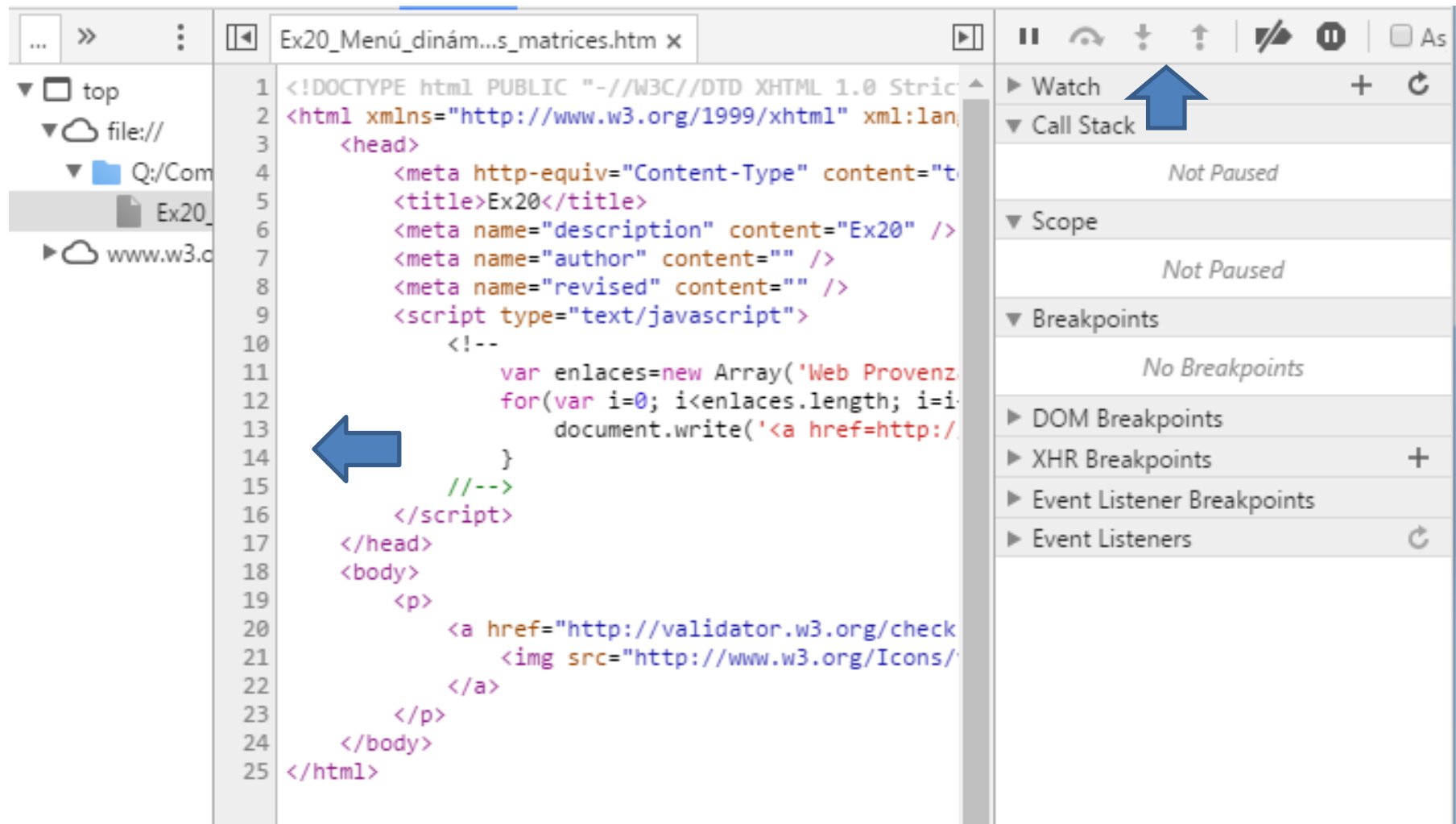


O bien botón derecho del ratón e Inspeccionar.

Nos fijamos en la parte de las herramientas “Source”.



” y pulsamos Ctrl+P para ver el código de la página activa o del módulo que seleccionemos en el árbol de la izquierda.

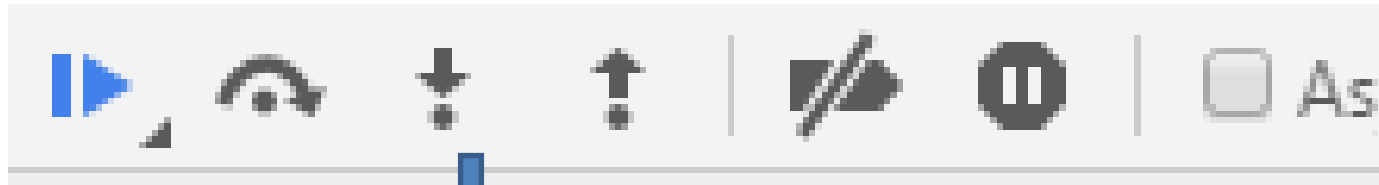


Click sobre un número de linea ponte un punto de parada.

 Reload, ejecuta nuevamente la página.

Los botones de la parte superior izquierda nos permiten parar , reiniciar y controlar la ejecución de l código.





Salta la línea



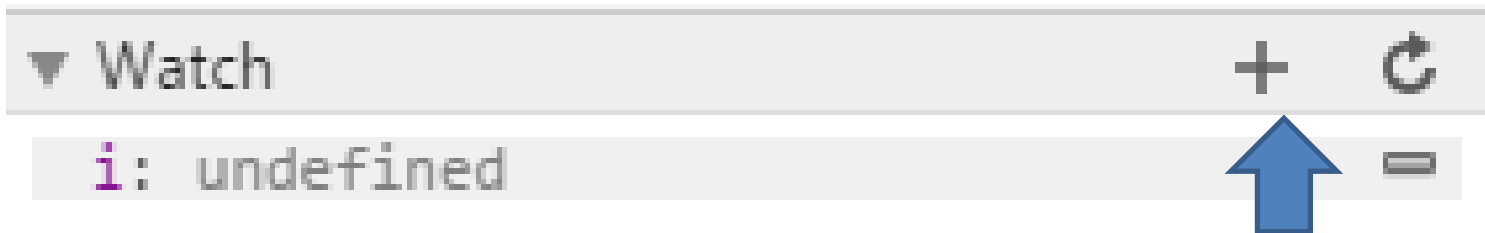
Entra a ejecutar la línea

Pausa y continua

```
Ex20_Menú_dinám...s_matrices.htm x
1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//E
2 <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="e
3 <head>
4     <meta http-equiv="Content-Type" content="text/
5     <title>Ex20</title>
6     <meta name="description" content="Ex20" />
7     <meta name="author" content="" />
8     <meta name="revised" content="" />
9     <script type="text/javascript">
10     <!--
11     var enlaces=new Array('Web Provenzana'
12     for (var i=0; i<enlaces.length; i=i+2){
13         undefined
14     }
15     //-->
16 </script>
17 </head>
18 <body>
19 <p>
20     <a href="http://validator.w3.org/check?uri
21     
10          <!--
11          var enlaces=new Array('Web Provenzana', 'www.pr
12          for(i=0;i<enlaces.length;i++)
13
14          }
15          //-->
16      </script>
17  </head>
18  <body>
19      <p>
20          <a href="http://www.provenzana.com">Web Provenzana</a>
21          <a href="http://www.google.es">Google</a>
22          <a href="http://www.xtec.cat">Xtec</a>
23          <a href="http://www.educalia.org">Educalia</a>
24      </p>
25  </body>
</html>
```

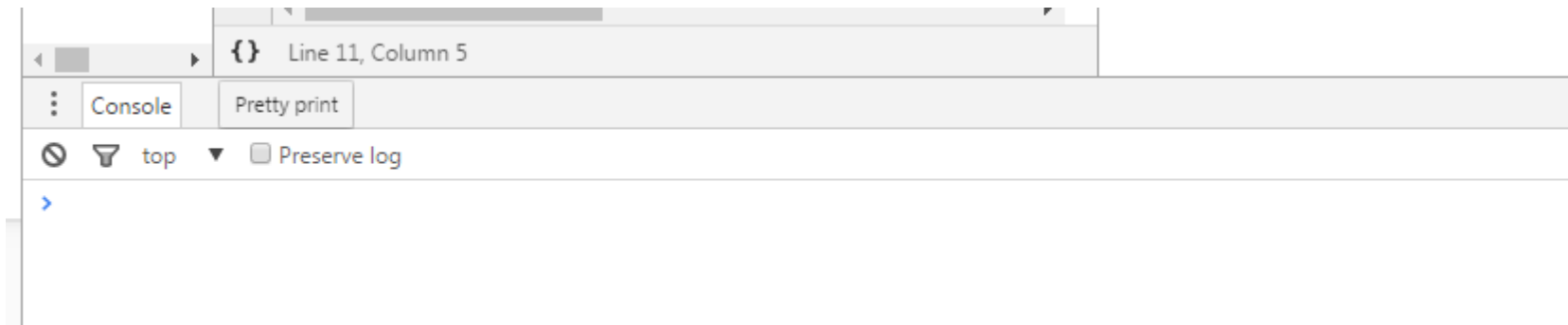
Array[8]


0: "Web Provenzana"
1: "www.provenzana.com"
2: "Google"
3: "ww.google.es"
4: "Xtec"
5: "www.xtec.cat"
6: "Educalia"
7: "www.educalia.org"
length: 8
▶ __proto__: Array[0]



Otra manera de inspeccionar una variable es añadir un punto de inspección o watch.

Tener la consola visible nos ayuda.



Recuerda recargar la página si quieres volver a ejecutar. 

FIN.