

# JavaScript



Es una de las múltiples aplicaciones que han surgido para extender las capacidades del Lenguaje HTML. JavaScript es un lenguaje script orientado a objetos.

JavaScript sirve principalmente para mejorar la gestión de la interfaz cliente/servidor.

# Diferencias con Java.

La principal diferencia entre JavaScript y Java, es que este último es un lenguaje de programación completo. Lo único que comparten es la misma sintaxis

JavaScript es un lenguaje que se integra directamente en páginas HTML. Tiene como características principales las siguientes:

Es interpretado por el cliente, no se genera ni código objeto ni ejecutable

Está basado en objetos. No es, como Java, un lenguaje de programación orientada a objetos (OOP).

Su código se integra en las páginas HTML, incluido en las propias páginas.

No es necesario declarar los tipos de variables que van a utilizarse, JavaScript realiza una conversión automática de tipos.

Las referencias a objetos se comprueban en tiempo de ejecución. Esto es consecuencia de que JavaScript no es un lenguaje compilado.

No puede escribir automáticamente al disco duro. Por eso decimos que JavaScript es un lenguaje seguro para el entorno de internet en el que se aplicará

# NORMAS DEL CODIGO EN JAVASCRIPT

1. Todo el código (sentencias) esta dentro de funciones.
2. Las funciones se desarrollan entre las etiquetas `<script>` y `</script>`.
3. Las etiquetas `"<script>"` deben colocarse entre las etiquetas `<head>` y `</head>`.
4. Las etiquetas `"<title>"` no pueden estar colocadas entre las de `"<script>"`.
5. La llamada a la función se hace a través de un evento de un elemento del documento.

# USO DE FUNCIONES

## SINTAXIS DEL DESARROLLO:

```
function nombre_funcion([var1,var2,varN])  
{  
  sentencia(s);  
}
```

## SINTAXIS DE LA LLAMADA:

```
<elemento evento=nombre_funcion([val1,val2,valN]);>  
  
nombre_funcion(valor1,valor2,valorN);
```

En el primero de los casos la llamada se realiza desde un elemento del documento. En el segundo caso la llamada se realiza desde el interior de otra función, que también es posible.

# LA VENTANA "ALERT"

## LA VENTANA "ALERT"

Se trata de una ventana estándar que usamos para mostrar información en pantalla. Se puede mostrar texto, variables y texto en conjunto con variables. El diseño de la ventana ya está definido lo único que podemos hacer es mostrar la información una o varias líneas. Su diseño y sintaxis es:

### SINTAXIS:

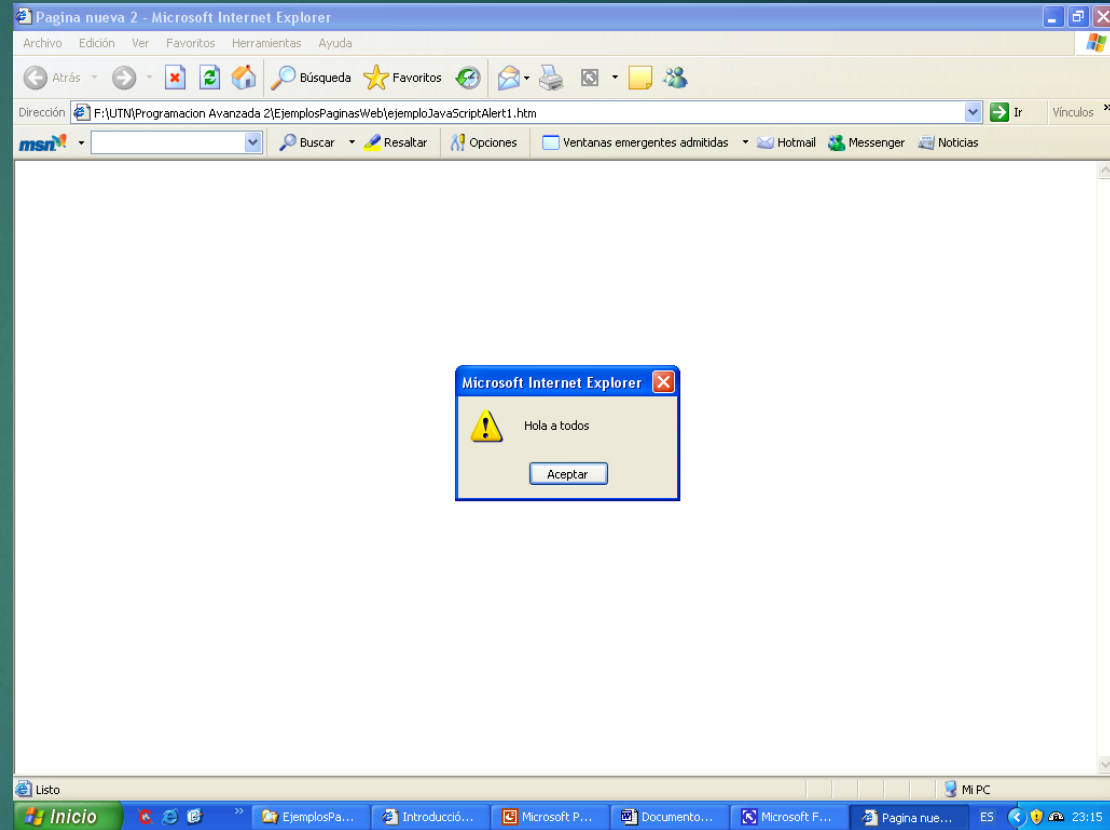
```
alert("texto de la ventana");
```

```
alert(variable);
```

```
alert("texto"+variable);
```

# Primer Programa

```
<html>
<head>
  <script>
    function hola()
    {
      alert("Hola a todos");
    }
  </script>
<title>Leonel</title>
</head>
<body onLoad=hola();>
</body>
</html>
```



# Eventos

- Un evento es un mecanismo por el cual podemos detectar las acciones que realiza el usuario y llamar automáticamente a la función que tengamos asociada.
- Desde esta función realizaremos las acciones que tengamos desarrolladas.

## SINTAXIS:

```
<elemento nombre_evento=nombre_funcion([parametros]) ;>
```



## EVENTO:SE PRODUCE AL

**onLoad:** Terminar de cargar una página o frame (entrar).

**onUnLoad:** Descargar una página o frame (salir).

**onAbort:** Abortar la carga de una página.

**onError:** Producirse algún error en la carga de la página.

**onMouseOver:** Pasar el ratón por encima de un enlace, area o frame.

**onMouseOut:** Dejar de estar el ratón encima de un enlace, area o frame.

**onMouseMove:** Mover el ratón por el documento.

**onKeyUp:** Presionar una tecla.

**onClick:** Hacer click con el ratón.

**onResize:** Dimensionar la ventana del navegador.

**onMove:** Mover la ventana del navegador.

**onChange:** Modificar texto en un control de edición. Sucede al perder el foco.

**onSelect:** Seleccionar texto en un control de edición.

**onFocus:** Situar el foco en un control.

**onBlur:** Perder el foco un control.

**onSubmit:** Enviar un formulario.

**onReset:** Inicializar un formulario.



# Variables y constantes

Espacio de memoria con un nombre, reservado para guardar información mientras la página este cargada.

## **Nombre de una variable:**

- ▶ No pueden contener espacios.
- ▶ Distingue entre mayúsculas y minúsculas.
- ▶ No pueden contener acentos, puntos o cualquier signo gramatical.
- ▶ No pueden comenzar con un dígito ni contener la letra "ñ".
- ▶ Nombre único y exclusivo para cada variable salvo que estén es 2 funciones distintas.

# Variables

El tipo de variable es asignado automáticamente por JavaScript. Depende del primer valor que se guarde en la variable. Por tanto los tipos de variable existentes son:

- ▶ numérica: Cualquier tipo numérico
- ▶ Boolean: True o False
- ▶ String: Texto o letra.

Otro aspecto importante, es la conversión de datos, que en JavaScript es automática. Transforma los datos de todas las variables en una expresión según el tipo de la primera variable. No es muy segura y puede acarrear muchos problemas.

## EJEMPLO:

```
num1="12";  
num2=10;  
x=num1+num2; // daría como resultado 1210.  
y=num2+num1; // daría como resultado 22.
```

# Variables

Para evitar problemas en las conversiones, se pueden utilizar métodos ya implementados que realizan la conversión de una manera más segura.

## **TIPO DE CONVERSION SINTAXIS**


De texto a numero:

```
enterovar_numerica=parseInt(var_texto)
```

De texto a coma flotante:

```
(decimal)var_numerica=parseFloat(var_texto)
```

De numérica a texto: Es automática sin peligro.



```
<html>
<head>
  <script>
    var global=100;
    function uno()
    {
      var local_uno=1;
      alert("Global " +global +" Local " +local_uno);
      dos();
    }
    function dos()
    {
      var local_dos=2;
      alert("Global " +global +" Local " +local_dos);
    }
  </script>
  <title>Leonel</title>
</head>
<body onLoad=uno();>
</body>
</html>
```

# Operadores

JavaScript define TRES tipos de operadores: aritméticos, relacionales y lógicos. También hay definido un operador para realizar determinadas tareas, como las asignaciones.

## ASIGNACION (=)

En JavaScript se puede utilizar el operador de asignación en cualquier expresión válida. Solo con utilizar un signo de igualdad se realiza la asignación. El operador destino (parte izquierda) debe ser siempre una variable, mientras que en la parte derecha puede ser cualquier expresión válida. Es posible realizar asignaciones múltiples, igualar variables entre si y a un valor.

### SINTAXIS:

`variable=valor;`

`variable=variable1;`

`variable=variable1=variable2=variableN=valor;`

# Operadores Aritméticos

## ARITMÉTICOS

Pueden aplicarse a todo tipo de expresiones. Son utilizados para realizar operaciones matemáticas sencillas, aunque uniéndolos se puede realizar cualquier tipo de operaciones. En la siguiente tabla se muestran todos los operadores aritméticos.

### OPERADOR DESCRIPCIÓN

- Resta

+ Suma

\* Multiplica

/ Divide

% Resto de una división

-- Decremento en 1

++ Incrementa en 1

vari+ = valor Incrementa el valor de vari

vari - = valor Decrementa el valor de vari.

vari \* = valor Multiplica el valor de vari.



# OPERADORES LÓGICOS Y RELACIONALES

- ▶ < Menor que
- ▶ > Mayor que
- ▶ <= Menor o igual
- ▶ >= Mayor o igual
- ▶ == Igual
- ▶ != Distinto

✓ && Y (AND)

✓ || O (OR)

✓ ! NO (NOT)



# INTRODUCCIÓN DE DATOS

JavaScript permite interactuar al usuario por medio de la introducción de datos. La introducción de datos se puede realizar por medio de la ventana prompt o utilizando controles como cajas de texto.

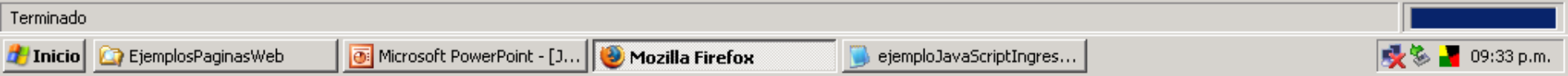
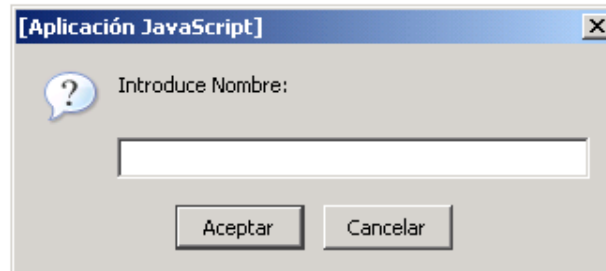
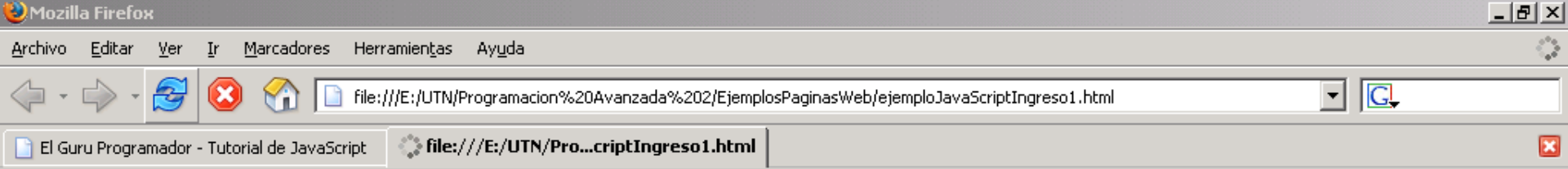
## VENTANA PROMPT:

```
vari=prompt("Texto de la  
ventana","valor inicial caja");
```

Al pulsar el botón aceptar, el contenido de la caja pasa a vari. Si se pulsa el botón cancelar, el contenido de la caja se pierde y vari queda con valor null.

## EJEMPLO:

```
<html><head>  
<script>  
    function valor()  
    {  
        var nombre;  
        nombre=prompt("Introduce  
Nombre:", "");  
        alert("hola "+nombre);  
    }  
</script>  
</head>  
<body onload=valor();>  
</body></html>
```

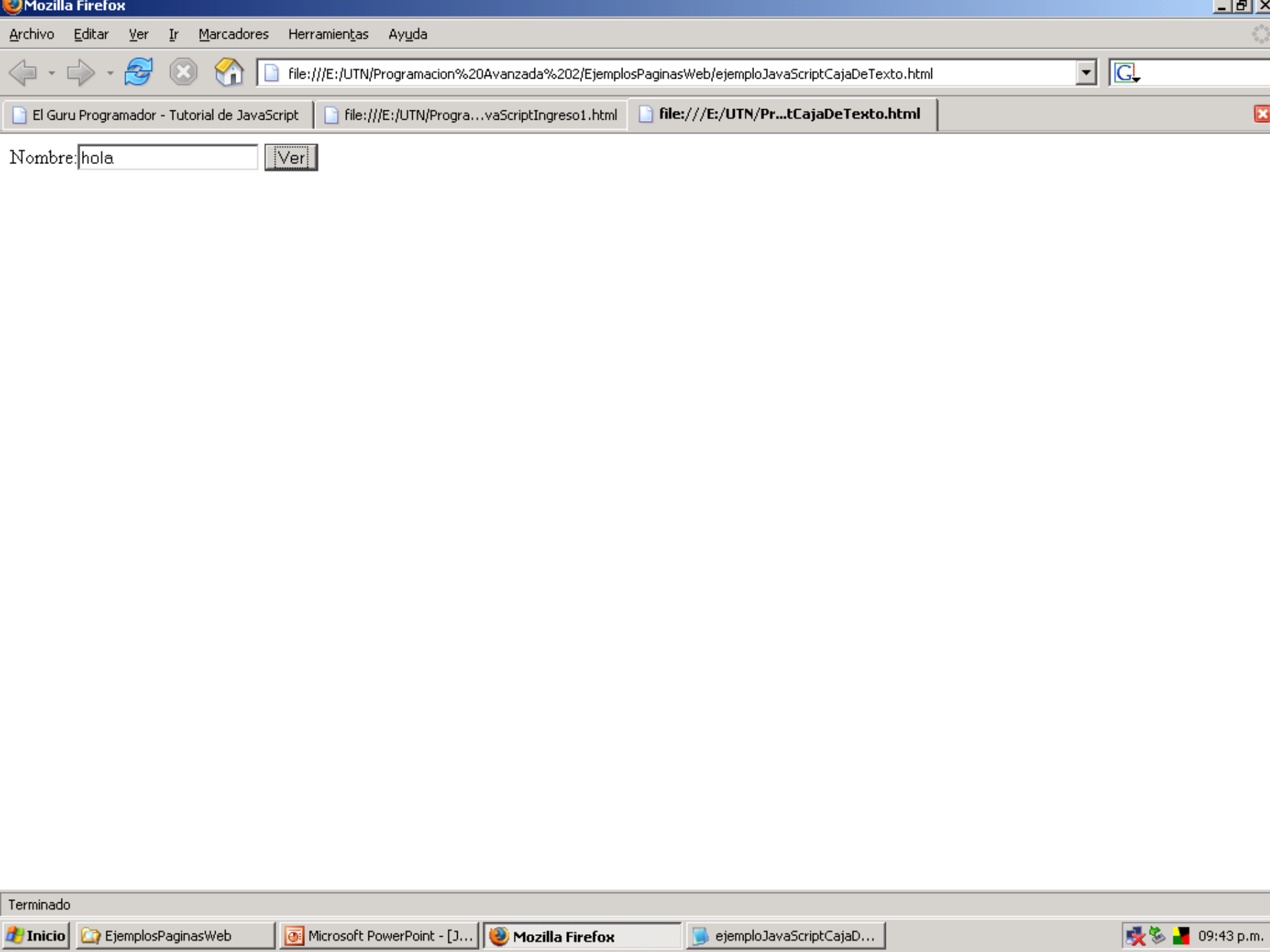


# CAJA DE TEXTO:

Otro mecanismo por el cual se pueden introducir datos, es mediante las cajas de texto. Todo el trabajo con las cajas, esta basado en funciones y métodos ya implementadas en JavaScript.

- ▶ `variable=nombre_caja.value;` Guarda el contenido de la caja
- ▶ `nombre_caja.value=valor o variable;` Muestra en la caja el valor
- ▶ `nombre_caja.value=""`; Limpia el contenido de la caja
- ▶ `nombre_caja.sefocus();` Envía el foco a la caja

```
<html><head><script>
function muestra(cnombre)
{
var nombre=cnombre.value;
alert("Eres "+nombre);
cnombre.value="";
cnombre.focus();
}
</script></head><body>
<form>
Nombre:<input type="text"
name="cnombre" size="20">
<input type="button" value="Ver"
onClick=muestra(this.form.cnombre);>
</form></body></html>
```



# SENTENCIAS DE CONTROL

## IF-ELSE:

La ejecución atraviesa un conjunto de estados boolean que determinan que bloques de código se ejecutan. Con la utilización de esta sentencia nunca se realizarán ambos bloques de código.


**SINTAXIS:** En caso de ser una sola sentencia por parte las llaves son opcionales.

```
if (expresion-booleana)
{
    Sentencia(s);
}
[else]
{
    Sentencia(s);
}
```

La cláusula else es opcional. La expresión puede ser de cualquier tipo y más de una (siempre que se unan mediante operadores lógicos). Otra opción posible es la utilización de if anidados, es decir unos dentro de otros compartiendo la cláusula else.

## Otro ejemplo

```
<html>
<head>
<script>
function pasa(secre,caja)
{
    var temporal;
    temporal=secre.value;
    caja.value=temporal;
}
</script>
</head>
<body>
<form>
CONTRASEÑA:
<input type="password" name="secre" size="5" onKeyUp=
    pasa(secre,caja) ;>
<br>
NORMAL:
<input type="text" name="caja" size="5">
</form>
</body>
</html>
```



```
<html>
  <head>
    <script>
      function ver()
      {
        var edad=parseInt(cedad.value);

        if(edad<=18)
          alert("No tienes acceso\nDebes tener
18");
        else
          alert("Bienvenido a la pagina");
      }
    </script>
    <title>Ejemplo if</title>
  </head>
  <body>
    Edad:
    <input type="text" name="cedad" size="3"
onBlur=ver();>
  </body>
</html>
```



# SWITCH:

Es una sentencia muy similar a if-else. Si los valores con los que se compara son números se los pone directamente, pero si es texto se debe encerrar entre comillas. La sintaxis de esta sentencia es:

```
switch (expresión)
{
    case constante1:
        sentencia(s);
        break;
    case constante2:
        sentencia(s);
        break;
    case constante3:
        sentencia(s);
        break;
    case constanteN:
        sentencia(s);
        break;
    [default:]
        sentencia(s);
}
```

El valor de la expresión se compara con cada una de las constantes de la sentencia case, si coincide alguno, se ejecuta el código que le sigue, y cuando ejecuta break sale de este bloque hasta salir del switch. Si ninguno coincide se realiza el bloque default (opcional), si no lo hay no se ejecuta nada.

```
<html>
  <head>
    <script>
      function espe()
      {
        var tipo=cespe.value;
        switch(tipo)
        {
          case "humano":
            alert("Eres un Humano");
            break;
          case "planta":
            alert("Eres un Vegetal");
            break;
          case "animal":
            alert("Eres del reino Animal");
            break;
          default:
            alert("Especie Desconocida");
            break;
        }
      }
    </script>
  </head>
  <body>
    ESPECIE:<input type="text" name="cespe" size="20" onBlur= espe() ;>
  </body>
</html>
```

# WHILE:

Ejecuta repetidamente el mismo bloque de código hasta que se cumpla una condición de terminación. Hay cuatro partes en todos los bucles. Inicialización, cuerpo, iteración y condición.

## SINTAXIS:

```
[inicialización;]  
while(condicion[es])  
{  
    cuerpo;  
    [iteración;]  
}
```

## DO..WHILE:

Es lo mismo que en el caso anterior pero aquí como mínimo siempre se ejecutará el cuerpo del bucle una vez, en el tipo de bucle anterior es posible que no se ejecute ni una sola vez el contenido de este.

### SINTAXIS:

```
[inicialización;]  
do  
{  
    cuerpo;  
    [iteración;]  
}while(condición);
```

## FOR:

Realiza las mismas operaciones que en los casos anteriores pero la sintaxis es una forma compacta. Normalmente la condición para terminar es de tipo numérico. La iteración puede ser cualquier expresión matemática válida. Si de los 3 términos que necesita no se pone ninguno se convierte en un bucle infinito.

**SINTAXIS:** En caso de ser una sola sentencia por parte las llaves son opcionales.

```
for (inicio;cond_fin;iteración)
{
    sentencia(S);
}
```

# CONFIRM:

## SINTAXIS:

```
var_boolean=confirm("texto de la ventana");
```

### **BOTON PULSADO**


### **VALOR De RETORNO**

ACEPTAR

true

CANCELAR

false



```
<html>
  <head>
    <script>
      function confirma()
      {
        var respuesta=confirm("Pulsa un botón");
        if (respuesta==true)
          alert("Has pulsado ACEPTAR");
        else
          alert("Has pulsado CANCELAR");
      }
    </script>
  </head>
  <body onLoad= confirma() ;>
  </body>
</html>
```



# PASO DE PARÁMETROS A FUNCIONES


Es el paso de información (parámetros) a una función. Cuando se realiza la llamada hay que indicar entre los paréntesis los valores que se van a enviar.

## SINTAXIS DE LA LLAMADA:

```
<elemento evento=nombre_funcion(valor1,valor2,valorN);>  
nombre_funcion(valor1,valor2,valorN);
```

## SINTAXIS DEL DESARROLLO:

```
function nombre_funcion(var1,var2,varN)  
{  
    sentencia(s);  
}
```



```
<html>
<head>
<script>
    function opt(valor)
    {
        if(valor==1)
            alert("Vas a ir a uno");
        else
            alert("Vas a ir a dos");
    }
</script>
<title>Ejemplo de funciones</title>
</head>

<body>
    <a href="Uno.htm" onMouseOver = opt(1); >ir a uno</a>
    <a href="Dos.htm" onMouseOver = opt(2); >ir a dos</a>
</body>
</html>
```

```
<html>
<head>
```

```
  <script>
```

```
    function opt(valor)
```

```
    {
```

```
      var cadena;
      switch(valor)
```

```
      {
```

```
        case 1:
```

```
          cadena="uno"; break;
```

```
        case 2:
```

```
          cadena="dos"; break;
```

```
        case 3:
```

```
          cadena="tres"; break;
```

```
        case 4:
```

```
          cadena="cuatro"; break;
```

```
      }
```

```
      alert("Vinculo " +cadena);
```

```
    }
```

```
  </script>
```

```
<title>Ejemplo de funciones</title>
```

```
</head>
```

```
<body><form>
```

```
<a href="Uno.htm" onMouseOver=opt(1);>uno</a>
```

```
<a href="Dos.htm" onMouseOver=opt(2);>dos</a>
```

```
<a href="Tres.htm" onMouseOver=opt(3);>tres</a>
```

```
<a href="Cuatro.htm" onMouseOver=opt(4);>cuatro</a>
```

```
</form></body>
```

```
</html>
```

# MATRICES (Arrays)

Una matriz es una colección de variables del mismo tipo que tiene un nombre común. A un elemento específico de una matriz se accede mediante su índice. Todos los arrays tienen como primer índice el valor 0. Hay que tener muy presente NO rebasar el valor del último índice.

## SINTAXIS:

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

## INICIALIZACIÓN DE UN ELEMENTO:

```
nombre_array[indice]=valor;
```

## UTILIZACIÓN DE ELEMENTOS:

*Casi como si se tratase de una variable normal.*

```
nombre_array[indice];
```

# Propiedades de las funciones.

JavaScript asocia a cada función dos propiedades; *arguments* y *caller*. Estas propiedades permiten respectivamente la gestión de los parámetros opcionales y la identificación de la función que llama.

## La propiedad *caller*.

Muestra el nombre de la función que llama, por lo tanto, esta propiedad devolverá una cadena de caracteres.

## La propiedad *arguments*.

Es un array que contiene los parámetros que le son pasados a la función. Por ejemplo, la función *suma* definida como:

```
function Suma(x)
{
    var sumandos = suma.arguments;
    var num_op = suma.arguments.length;
    var resultado = 0;
    for (var i=0; i<num_op; i++)
        resultado += sumandos[i];
    return resultado;
}
```

Vemos como esta función permite calcular la suma de los números pasados como argumentos. Así, *Suma(4,5,7)* devuelve 16 y *Suma(56)* devuelve 56.

# Las funciones predefinidas por el lenguaje.

## La función eval.

La función eval tiene como argumento una expresión y devuelve el valor de la misma. Esta función resulta útil para evaluar una cadena de caracteres que representa una expresión numérica. La edición efectuada mediante un campo de formulario es una cadena de caracteres que a veces es necesario convertir en valor numérico. El código siguiente ilustra este ejemplo permitiendo al usuario introducir una expresión numérica y visualiza a continuación el valor de la expresión.

```
<SCRIPT>
```

```
function calcula(obj)
```

```
{
```

```
    obj.result.value = eval(obj.expr.value)
```

```
} </SCRIPT>
```

```
<FORM NAME="evalua">
```

```
Introducir expresión:
```

```
<INPUT TYPE="text" NAME="expr" SIZE=20> <br>
```

```
Resultado: <INPUT TYPE="text" NAME="result" SIZE=20><br>
```

```
<INPUT TYPE="button" VALUE="evalua" onClick="calcula(this.form)">
```

```
</FORM>
```



## La función isNaN.

Función que comprueba si el valor pasado por parámetros es numérico o no. El resultado de esta función es un booleano. Es decir, evalúa un argumento para ver si es NaN: *Not Number*.

isNaN(valor de entrada)

### Ejemplo

```
<SCRIPT>
```

```
function Comprueba(form)
{
    var number = parseFloat(form.valor.value);
    if (isNaN(number)==true)
        alert("No es numérico");
    else form.valor.value = number;
        alert("Es numérico");
}
```

```
</SCRIPT> ...
```

```
<form>
```

```
Introducir un valor numérico: <input type="text" name="valor"><br>
    <input type="button" value=" Comprobar "
    onClick="Comprueba(this.form)"> </form>
```