



JavaScript

Lenguaje de programación que se utiliza principalmente para crear páginas web dinámicas (con efectos, animaciones e interacción con el usuario).

Funciona del lado del cliente.

Para más info, ir a Libros Web:

<http://librosweb.es/javascript/>

O a JavaScript Ya:

<http://www.javascriptya.com.ar/>

INCLUIR JAVASCRIPT EN DOCUMENTOS HTML:

Al igual que CSS, se puede incluir de 3 formas:

1) En línea:

```
<p onclick="alert('Un mensaje de prueba')">Un párrafo de texto.</p>
```

2) Forma interna embebida:

```
<head>
  <script type="text/javascript">
    alert("Un mensaje de prueba");
  </script>
</head>
```

3) Archivo externo:

pagina.html

```
<!DOCTYPE html>
```

```
<html lang="es">
```

```
  <head>
```

```
    <meta charset="utf-8"/>
```

```
    <title>Incluir JavaScript archivo externo</title>
```

```
    <!--Scripts Aca -->
```

```
    <script type="text/javascript" src="pagina.js">
```

```
  </script>
```

```
  </head>
```

```
  <body>
```

```
  </body>
```

```
</html>
```

pagina.js

```
A=7
```

```
C=10
```

```
document.write(A+C);
```

Otro ejemplo:

```
var nombre='Ricardo';  
  
alert('Hola'+nombre);
```

alert() → imprime en una ventana de alerta.

VARIABLES:

Las variables pueden ser de tipo:

- Enteros
- Flotantes
- Cadenas de texto
- Elemento de html
- Objetos

Ej:

```
var iva = 16;      // variable tipo entero  
  
var total = 234.65; // variable tipo decimal  
  
var mensaje = "Bienvenido a nuestro sitio web"; //cadena de texto
```

var d; → se usa una vez para declarar la variable.

Si se define fuera de una función, es una variable global.

JavaScript divide los distintos tipos de variables en dos grupos: tipos primitivos y tipos de referencia o clases.

Tipos primitivos: hay cinco tipos primitivos: undefined, null, boolean, number y string. Además de estos tipos, JavaScript define el operador **typeof** para averiguar el tipo de una variable.

```
var variable1 = 7;  
  
typeof variable1; // "number"  
  
var variable2 = "hola mundo";  
  
typeof variable2; // "string"
```

Tipos de referencia o clases: parecidos a las clases de otros lenguajes de programación:

```
var cadena_texto = new String("hola mundo");  
  
var fecha = new Date(2009, 11, 25); // variable1 = 25 diciembre de 2009  
  
var numero = new Object(5);          // numero es de tipo Number  
  
var cadena = new Object("hola mundo"); // cadena es de tipo String  
  
var conectado = new Object(false);   // conectado es de tipo Boolean  
  
var booleano = new Boolean(false);  
  
var numero2 = new Number(3.141592);
```

instanceof → sirve para determinar la clase concreta de un objeto.

```
var variable1 = new String("hola mundo");  
  
typeof variable1;          // devuelve "object"  
  
variable1 instanceof String; // devuelve true
```

Nota: Para mas info, ver el capítulo 2.4: tipos de Variables, de Ajax:

http://librosweb.es/ajax/capitulo_2/tipos_de_variables.html

OPERADOR TERNARIO:

```
var aux=true;  
  
var asignada=aux?'hola':'adiós';  
  
alert(asignada);
```

aux? → Pregunta si aux es verdadero (true), si lo es, a la variable asignada le ingresa “hola”. Y si es falsa le asigna: “adiós”.

ARRAYS:

```
var a=[ ];  
  
var a=new Array();  
  
a.push(5);  
  
a.push("Colombia");
```

```
a.push(9.81);
```

```
var valores=[51.9,"Hola",7];  
for(var i=0;i<valores.length;++i){  
    document.write(valores[i]+"<br>");  
}
```

En JavaScript, se pueden incluir etiquetas Html o estilos CSS.

CICLO FOR IN:

Este ciclo se vera más adelante en JavaScript Avanzado, con Objetos.

```
for(índice in array){...}
```

```
var días=["Lunes","Martes","Miércoles","Jueves","Viernes","Sábado", "Domingo"];  
for (i in dias){  
    alert(días[i]);  
}
```

FUNCIONES:

```
function factorial(num){  
    var fac=1;  
    for(num;num>0;num--)  
        fac=fac*num;  
    return fac;  
}
```

```
var resultado= factorial(3);
```

```
alert(resultado);
```

//Otro tipo de llamado a la función, e imprime al mismo tiempo:

```
alert(factorial(3));
```

Otra forma de declarar la función es:

```
var factorial=function (num){  
    var fac=1;  
    for(num;num>0;num--)  
        fac=fac*num;  
    alert(fac);  
}  
  
factorial(5);
```

Esta función no retorna nada, y su llamado se hace con la variable.

Función Anónima:

```
function(){  
    //instrucciones que se quieren ejecutar.  
}
```

Funciones Útiles para Cadenas de Texto:

.length → calcula la longitud de una cadena de texto.

```
var mensaje="Hola Mundo";  
var num_letras=mensaje.length;  
  
//num_letras=10
```

.concat() → concatena igual que el "+"

```
mensaje1="Hola";  
mensaje2=mensaje1.concat(" mundo");  
  
//mensaje2="Hola mundo"
```

.toUpperCase() → pasa todo a mayúsculas.

```
mensaje2=mensaje1.toUpperCase(); //mensaje2="HOLA"
```

.toLowerCase() → pasa a minúsculas.

.charAt(posicion) → obtiene el carácter que se encuentra en la posición indicada.

```
var letra = mensaje1.charAt(0); // letra = 'H'
```

```
letra = mensaje1.charAt(2); // letra = 'l'
```

.indexOf(caracter) → indica la posición del carácter dado. Devuelve la primera posición empezando a contar desde la izquierda.

Si la cadena no tiene el carácter devuelve un -1

```
var mensaje1 = "Hola";
```

```
var posicion = mensaje1.indexOf('a'); // posicion = 3
```

```
posicion = mensaje1.indexOf('b'); // posicion = -1
```

.lastIndexOf(caracter) → análoga a la función indexOf(), devuelve la última posición del carácter.

.substring(inicio,final) → extrae un pedazo de la cadena. El parámetro “final” es opcional.

```
var mensaje = "Hola Mundo";
```

```
var porcion = mensaje.substring(2); // porcion = "la Mundo"
```

```
porcion = mensaje.substring(1,8); // porcion = "ola Mun"
```

```
var porcion = mensaje.substring(-2); // porcion = "Hola Mundo"
```

.split(separador) → convierte una cadena de texto en un array de cadenas de texto.

```
var mensaje = "Hola pollo, perro conejo ";
```

```
var palabras = mensaje.split(" ");
```

```
// palabras = ["Hola", "pollo,", "perro", "conejo"];
```

```
var palabra = "Hola";
```

```
var letras = palabra.split(""); // letras = ["H", "o", "l", "a"]
```

Funciones Útiles para Arrays:

.length → indica la cantidad de elementos.

.concat() → concatena elementos de varios arrays.


```
var array1 = [1, 2, 3];
```

```
var array2 = array1.concat(4, 5, 6); // array2 = [1, 2, 3, 4, 5, 6]
```

.join(separador) → función contraria a **.split()**. Une los elementos.

```
var array = ["hola", "mundo"];
```

```
var mensaje = array.join(""); // mensaje = "holamundo"
```

```
mensaje = array.join(" "); // mensaje = "hola mundo"
```

.pop() → elimina y devuelve el último elemento.

```
var array = [1, 2, 3];
```

```
var ultimo = array.pop();
```

```
// array = [1, 2] ultimo=3
```

.push() → añade al final.

```
array.push(4);
```

```
// array = [1, 2, 4]
```

.shift() → elimina y devuelve el primer elemento.

```
var primero = array.shift();
```

```
// array = [2, 4] primero=1
```

.unshift() → añade al principio.

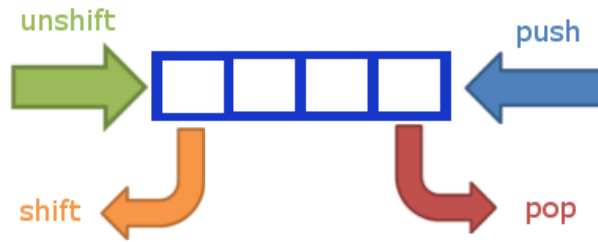
```
array.unshift(0);
```

```
// array = [0, 2, 4]
```

.reverse() → var array = [1, 2, 3];

```
array.reverse();
```

```
// array = [3, 2, 1]
```



Funciones Útiles para Números:

NaN → (Not a Number), indica un valor numérico no definido (EJ: 0/0)

```
var num1=0;
```

```
var num2=0;
```

```
alert(num1/num2); // se muestra el valor NaN
```

isNaN() → permite proteger la aplicación de posibles valores numéricos no definidos.

```
if(isNaN(num1/2)){  
    alert("La división no está definida para los números indicados");  
}else{  
    alert("La división es igual a "+num1/num2);  
}
```

Infinity → hace referencia al valor numérico infinito.

```
var num1=10;
```

```
var num2=0;
```

```
alert(num1/num2); // se muestra el valor Infinity.
```

toFixed(digitos) → devuelve el # con los decimales indicados, y realiza los redondeos necesarios.

```
var numero1 = 4564.34567;
```

```
numero1.toFixed(2); // 4564.35
```

```
numero1.toFixed(6); // 4564.345670
```

numero1.toFixed(); // 4564

DOM:

Document Object Model.

Los navegadores, cuando abrimos una pág. web, automáticamente generan un árbol de nodos.

DOM, transforma todos los documentos XHTML en arboles de nodos.

Ej:

```
<!DOCTYPE html>
```

```
<html>
```

```
  <head>
```

```
    <meta charset="utf-8"/>
```

```
    <title>Página Sencilla</title>
```

```
  </head>
```

```
  <body>
```

```
    <h1>Ejercicio</h1>
```

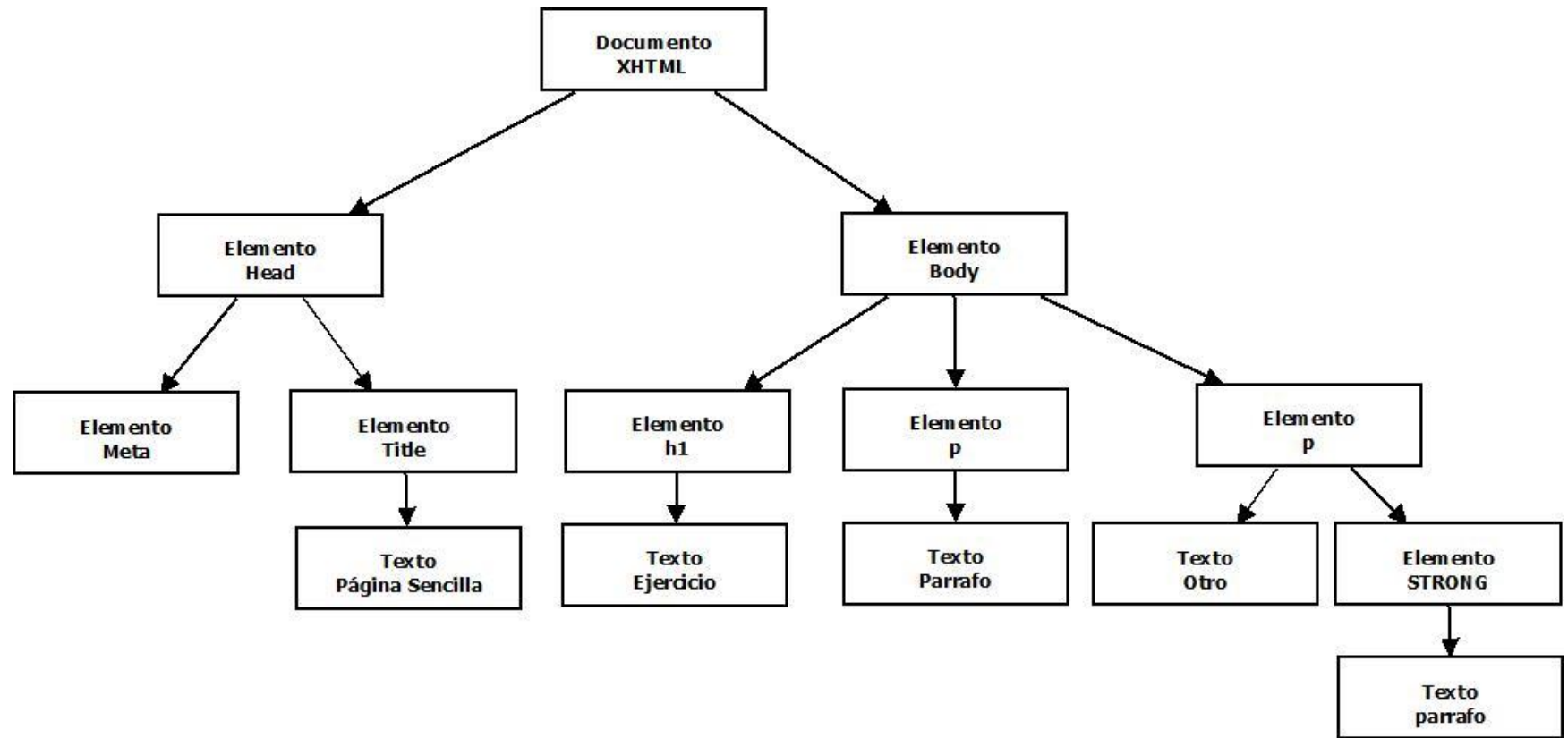
```
    <p>Parrafo</p>
```

```
    <p>Otro <strong>parrafo</strong></p>
```

```
  </body>
```

```
</html>
```

Se transforma en el siguiente árbol de nodos:



- Cada rectángulo es un nodo.
- El nodo raíz, siempre es documento.
- Cada etiqueta XHTML, se transforma en un nodo de tipo "elemento".

La transformación de una pág. en árbol de nodos sigue las siguientes reglas:

- (1) Las etiquetas XHTML, se transforman en 2 nodos: el primero es la propia etiqueta y el segundo nodo es hijo del primer nodo y consiste en el contenido textual de la etiqueta.
 - (2) Si una etiqueta XHTML, se encuentra dentro de otra, se sigue el procedimiento anterior, pero los nodos generados serán nodos hijo de su etiqueta padre.
-

Acceso a los Nodos:

Se hace a través de funciones del DOM; y solo cuando la pág. a cargado completamente.

.getElementsByTagName() → obtiene todos los elementos de la pág. cuya etiqueta sea igual a la que se le pasa a la función.

```
var parrafos=document.getElementsByTagName("p");
```

//saca todos los párrafos

Se puede usar de forma recursiva:

```
var parrafos=document.getElementsByTagName("p");
```

```
var primerParrafo=parrafos[0];
```

```
var enlaces=primerParrafo.getElementsByTagName("a");
```

//saca un array con todos los enlaces.

.getElementByName() → busca solo el elemento cuyo nombre sea igual al indicado.

```
var parrafoEspecial=document.getElementByName("especial");
```

```
<p name="prueba">...</p>
```

```
<p name="especial">...</p>
```

```
<p>...</p>
```

.getElementById() → como el id es único para c/elemento, devuelve solo el nodo deseado.

```
var cabecera=document.getElementById("cabecera");
```

```
<div id="cabecera">  
    ...  
</div>
```

Crear y Añadir Elementos (nodos):

Consta de 4 pasos:

- (1) Crear un nodo de tipo Element para representar el elemento.
- (2) Crear un nodo del tipo Text que represente el contenido del elemento.
- (3) Añadir el nodo Text, como hijo del nodo Element.
- (4) Añadir el nodo Element a la pág. en forma de hijo del nodo correspondiente al lugar donde se quiere insertar el elemento.

// (1) Crear nodo de tipo Element

```
var parrafo=document.createElement("p");
```

// (2) Crear nodo de tipo Text

```
var contenido=document.createTextNode("Hola Mundo!");
```

// (3) Añadir el nodo Text como hijo del nodo Element.

```
parrafo.appendChild(contenido);
```

// (4) Añadir el nodo Element como hijo de la pág.

```
document.body.appendChild(parrafo);
```

La creación de un elemento, implica usar las 3 funciones en negrilla:

```
.createElement(etiqueta);
```

```
.createTextNode(contenido);
```

```
.nodoPadre.appendChild(nodoHijo);
```

Eliminar Nodos:

```
var parrafo=document.getElementById("provisional");
```

```
párrafo.parentNode.removeChild(parrafo);
```

```
<p id="provisional">...</p>
```

.removeChild → se invoca desde el elemento padre del nodo.

Para acceder al padre de este elemento o cualquier otro se usa:

nodoHijo.**parentNode**

Nota: cuando se elimina un nodo, se eliminan todos sus hijos.

Acceso a los Atributos XHTML:

Cuando se accede a un nodo, también es posible cambiar y acceder a sus atributos y propiedades (incluidas propiedades de CSS).

Ej: obtener la dirección URL de un enlace (etiqueta <a>).

```
var enlace=document.getElementById("enlace");  
alert(enlace.href); //muestra http://www. .com
```

```
<a id="enlace" href="http://www. .com">Enlace</a>
```

Para obtener por ejemplo el atributo id, se utilizaría enlace.id.

Para obtener una propiedad **CSS**, se utiliza el atributo style.

Ej: obtener el valor de la propiedad margen (margin) de la imagen.

```
var imagen=document.getElementById("imagen");  
alert(imagen.style.margin);
```

```

```

Si el nombre de una propiedad CSS, es compuesto, se accede a su valor modificando ligeramente su nombre.

Ej: var parrafo= document.getElementById("parrafo");

```
alert(párrafo.style.fontWeight); // muestra "bold"
```

```
<p id="parrafo" style="Font-weight:bold;">...</p>
```

La transformación del nombre de las propiedades CSS, consiste en eliminar el guion medio (-) y escribir en mayúscula la letra siguiente a c/guion medio.

Ej: font-weight → fontWeight
line-height → lineHeight
border-top-style → borderTopStyle

En JavaScript la palabra Class está reservada, por lo que se usa **className**:

```
var parrafo=document.getElementById("parrafo");  
alert(párrafo.className); //muestra "normal"
```

```
<p id="parrafo" class="normal">...</p>
```

EJERCICIO DOM:

Dinámicamente, cambia el estilo del título y los párrafos iniciales, adicionalmente, , intercambia los párrafos iniciales, los oculta/muestra, agrega un párrafo al principio, y final del documento y elimina parrafos.

dom.html

```
<!DOCTYPE html>
```

```
<html>
```

```
  <head>
```

```
    <title>DOM</title>
```

```
    <script type="text/javascript" src="dom.js"></script>
```

```
  </head>
```

```
  <body>
```

```
    <h1 id="titulo">Ejercicios DOM parte 1</h1>
```

```
    <p>Dom, Modelo dinamico de objetos</p>
```

```
    <p>programacion <strong>web</strong></p>
```

```
    </br><span id="intercambiar">Intercambiar</span>
```

```
    </br><span id="ocultar">Esconder</span>
```

```
    </br><span id="mostrar">Ver</span>
```

```
    </br><span id="agregarFinal">Agregar Al Final</span>
```

```
    </br><span id="agregarInicio">Agregar Al Inicio</span>
```

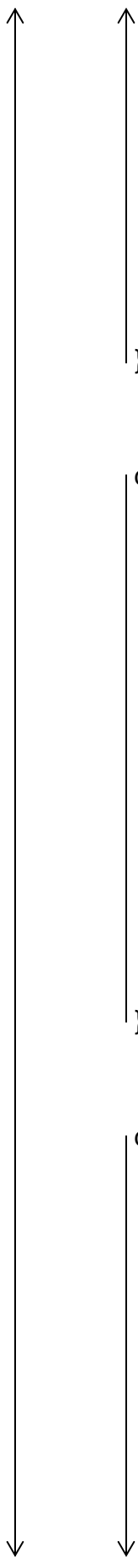
```
    </br><span id="agregarParrafo">Agregar un Parrafo</span>
```

```
    </br><span id="eliminarParrafo">Eliminar un Parrafo</span>
```

```
  </body>
```

```
</html>
```

```
window.onload=function(){  
    var contador=1;  
  
    var elemento=document.getElementById("titulo");  
    elemento.style.color="red";  
    var ps=document.getElementsByTagName("p");  
    for(var i=0;i<ps.length;++i){  
        ps[i].style.color="blue";  
    }  
    document.getElementById("intercambiar").onclick=function(event){  
        var temporal=ps[0].innerHTML;  
        ps[0].innerHTML=ps[1].innerHTML;  
        ps[1].innerHTML=temporal;  
    }  
  
    document.getElementById("ocultar").onclick=function(event){  
        for(var i=0;i<ps.length;++i){  
            ps[i].style.display="none";  
        }  
    }  
  
    document.getElementById("mostrar").onclick=function(event){  
        for(var i=0;i<ps.length;++i){  
            ps[i].style.display="block";  
        }  
    }  
  
    document.getElementById("agregarFinal").onmouseover=function(event){  
        var nuevo=document.createElement("div");  
        var contenido=document.createTextNode("Soy nuevo");  
        /*Agrega el contenido al final*/  
        nuevo.appendChild(contenido);  
        document.body.appendChild(nuevo);  
    }  
  
    document.getElementById("agregarInicio").onclick=function(event){  
        var nuevo=document.createElement("div");
```



```
var contenido=document.createTextNode("Me agregaron de
primeras");
/*Agrega el contenido al inicio*/
nuevo.appendChild(contenido);
/*Saca el primer elemento*/
var primero=document.body.childNodes[0];

document.body.insertBefore(nuevo,primero);
}
```

```
document.getElementById("agregarParrafo").onclick=function(event){

    /*Crea una nueva etiqueta <p>*/
    var nuevo=document.createElement("p");
    var contenido=document.createTextNode("Parrafo"+contador);
    ++contador;
    /*agrega contenido a la etiqueta*/
    nuevo.appendChild(contenido);
    /*saca todos los parrafos*/
    var ps=document.getElementsByTagName("p");
    /*selecciona el ultimo parrafo*/
    var ultimo=ps[ps.length-1];

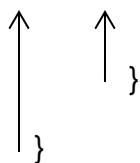
    /*Agrega antes del siguiente hijo del papa, osea la última etiqueta*/
    document.body.insertBefore(nuevo,ultimo.nextSibling);

}
```

```
document.getElementById("eliminarParrafo").onclick=function(event){

    /*saca todos los parrafos*/
    var ps=document.getElementsByTagName("p");
    /*selecciona el ultimo parrafo*/
    if(ps.length>0){
        var ultimo=ps[ps.length-1];
        /*elimina el último, diciéndole a la etiqueta padre, cualquiera
        que sea que borre al hijo, en este caso sí mismo*/
        ultimo.parentNode.removeChild(ultimo);
    }

}
```



nextSibling, previousSibling → se usan para acceder a nodos hermanos.

EVENTOS:

Cuando el usuario hace algún tipo de interacción con el contenido de la página, se crea un evento. Hay muchos tipos de eventos, como: hacer clic en un botón, pasar el mouse encima, arrastrar y soltar, presionar una tecla, etc.

Un evento en JavaScript es un objeto.

El nombre de cada evento se construye mediante el prefijo **on**, seguido del nombre en inglés de la acción asociada al evento.

Ej: onclick

La siguiente tabla resume los eventos más importantes definidos por JavaScript:

| 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 |

| Evento | Descripción | Elementos para los que está definido |
|-------------|---|--------------------------------------|
| | | y <body> |
| onkeyup | Soltar una tecla pulsada | Elementos de formulario y <body> |
| onload | La página se ha cargado completamente | <body> |
| 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 ratón | Todos los elementos |
| 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> |

Manejador de Eventos:

Se refiere a las funciones o código JavaScript que se define para cada evento. Los manejadores se pueden indicar de 3 formas:

(1) Manejadores como atributos de los elementos XHTML:

```

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

    Contenido del div...

</div>

```

this → hace referencia al elemento XHTML que ha provocado el evento.

(2) Manejadores como funciones JavaScript externas:

```

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

```

```

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

```

El inconveniente de usar funciones externas, es que toca pasar la variable **this** como parámetro a la función.

Se usan muchos “case”, debido a la interpretación de borderColor, por parte de los navegadores.

(3) Manejadores Semánticos:

Es la mejor forma, ya que no ensucia el código XHTML. Y dentro de las funciones externas asignadas sí se puede utilizar la variable **this** para referirse al elemento que provoca el evento.

Para usar este método la pág. se debe cargar completamente antes de que se puedan usar las funciones DOM.

Para asegurarse que el código JavaScript se ejecute después de cargar la pág. se usa:

```
    window.onload=function(){  
        ...  
    }
```

Así el manejador semántico quedaría:

//Función externa

```
function muestraMensaje(){  
    alert('Gracias por pinchar');  
}
```

//verificar que se carga la pág.

```
window.onload=function(){  
    //Asignar la función externa al elemento  
    document.getElementById("pinchable").onclick=muestraMensaje;  
}
```

//Elemento XHTML

```
<input id="pinchable" type="button" value="Pinchame y veras!"/>
```

La técnica de los manejadores semánticos consiste en:

- (A) Asignar un identificador único al elemento XHTML mediante el atributo id.
- (B) Crear una función de JavaScript encargada de manejar el evento.
- (C) Asignar la función externa al evento correspondiente en el elemento deseado.

document.getElementById("pinchable") → se obtiene el elemento.

document.getElementById("pinchable").onclick → Se usa la propiedad de elemento con el nombre del evento que se va a manejar (en este caso onclick).

document.getElementById("pinchable").onclick=muestraMensaje; → se asigna la función externa al evento.

Nota: Lo más importante (y la causa más común de errores) es indicar solamente el nombre de la función, es decir, prescindir de los paréntesis al asignar la función.

Si se añaden los paréntesis después del nombre de la función, en realidad se está ejecutando la función y guardando el valor devuelto por la función en la propiedad onclick de elemento.

// Asignar una función externa a un evento de un elemento

```
document.getElementById("pinchable").onclick = muestraMensaje;
```

// Ejecutar una función y guardar su resultado en una propiedad de un elemento

```
document.getElementById("pinchable").onclick = muestraMensaje();
```

Objeto Event:

Este objeto nos sirve para recuperar información del evento, como por ejemplo que tecla se presionó, etc.

El objeto lo crean automáticamente los navegadores, solo es necesario asignarle un nombre.

Para obtenerlo en cualquier navegador se usa:

```
function manejadorEventos(elEvento){  
    var evento=elEvento || window.event;  
}
```

window.event → forma en cómo se obtiene en IE.

Información sobre el Evento:


type → indica el tipo de evento producido, es útil cuando una misma función se usa para manejar varios eventos.

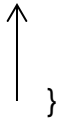
```
var tipo=evento.type;
```

type devuelve el tipo de evento producido, que es igual al nombre del evento pero sin el prefijo on.

Ej: resaltar una sección de contenidos al pasar el ratón por encima.

```
function resalta(elEvento){  
    var evento=elEvento || window.event;  
    switch(evento.type){  
        case 'mouseover': this.style.borderColor='black';  
                           break;  
        case 'mouseover': this.style.borderColor='black';  
                           break;  
    }  
}  
  
window.onload=function(){  
    document.getElementById("seccion").onmouseover=resalta;  
    document.getElementById("seccion").onmouseout=resalta;  
}
```





```
<div id="seccion" style="width:150px; height:60px; border:thin solid silver" >  
    Contenido del div...  
</div>
```

Eventos del Teclado:

Cuando se presiona una tecla se generan 3 eventos en el siguiente orden:

onkeydown → pulsar la tecla y mantenerla presionada.

onkeypress → la pulsación de la tecla.

onkeyup → soltar la tecla que estaba pulsada.

Para obtener en cualquier navegador el carácter correspondiente a la tecla pulsada:

```
function manejador(elEvento) {  
    var evento = elEvento || window.event;  
    var caracter = evento.charCode || evento.keyCode;  
    alert("El carácter pulsado es: " + String.fromCharCode(caracter));  
}
```

```
document.onkeypress = manejador;
```

Información Eventos del Ratón:

La forma de obtener las coordenadas de donde se hizo clic con el ratón, varían con respecto a la pantalla, la ventana del navegador, etc.

- Respecto a la ventana del navegador:



Se obtienen las coordenadas con las propiedades **clientX** y **clientY**.

```
function muestraInformacion(elEvento) {
    var evento = elEvento || window.event;
    var coordenadaX = evento.clientX;
    var coordenadaY = evento.clientY;

    alert("Has pulsado el ratón en la posición: " + coordenadaX + ", " +
    coordenadaY);
}

document.onclick = muestraInformacion;
```

- Respecto a la pantalla completa del pc.



screenX y screenY

- Respecto al origen de la página, es similar a: "Respecto a la ventana del navegador", pero se usa si se ha hecho scroll.

```
//Detectar si el navegador es Internet Explorer
```

```
var ie = navigator.userAgent.toLowerCase().indexOf('msie')!=-1;
```

```
if(ie) {  
    coordenadaX = evento.clientX + document.body.scrollLeft;  
    coordenadaY = evento.clientY + document.body.scrollTop;  
}  
else {  
    coordenadaX = evento.pageX;  
    coordenadaY = evento.pageY;  
}  
alert("Has pulsado el ratón en la posición: " + coordenadaX + ", " +  
    coordenadaY + " respecto de la página web");
```

UTILIDADES PARA FORMULARIOS:

C/u de los elementos de un formulario (cuadros de texto, botones, listas desplegables, etc.), tienen las siguientes propiedades:

type → indica el tipo de elemento.

- Para: <input> (text ,button, checkbox, etc.), coincide con el valor de su atributo type.
- Para: <select> (lista desplegable normal) su valor es select-one.
- Para listas donde se selecciona varios elementos es select-multiple.
- Para <textarea> el valor de type es textarea.

form → hace referencia directa al formulario al que pertenece el elemento. Para acceder al formulario de un elemento, se puede usar:

```
document.getElementById("id_del_elemento").form
```

name → obtiene el valor del atributo name de XHTML. Solo se puede leer el valor, no modificar.

value → permite leer y modificar el valor del atributo value de XHTML.

- **Obtener el valor de un cuadro de texto y un textarea:**

```
<input type="text" id="texto" />
```

```
var valor = document.getElementById("texto").value;
```

```
<textarea id="parrafo"></textarea>
```

```
var valor = document.getElementById("parrafo").value;
```

- **Radiobutton:**

.checked → devuelve true para el radiobutton seleccionado.

```
<input type="radio" value="si" name="pregunta" id="pregunta_si"/> SI
```

```
<input type="radio" value="no" name="pregunta" id="pregunta_no"/> NO
```

```
<input type="radio" value="nsnc" name="pregunta" id="pregunta_nsnc"/>  
No sabe no responde
```

Para determinar cuál radiobutton ha sido seleccionado:

```
var elementos = document.getElementsByName("pregunta");  
  
for(var i=0; i<elementos.length; i++) {  
    if(elementos[i].checked){  
        alert(" Elemento: " + elementos[i].value + "\n Seleccionado: " +  
            elementos[i].checked);  
    }  
}
```

- **Checkbox:**

```
<input type="checkbox" value="condiciones" name="condiciones"  
id="condiciones"/> He leído y acepto las condiciones
```

```
<input type="checkbox" value="privacidad" name="privacidad" id="privacidad"/>  
He leído la política de privacidad
```

```
var elemento = document.getElementById("condiciones");
```

```
alert(" Elemento: " + elemento.value + "\n Seleccionado: " +  
    elemento.checked);
```

```
elemento = document.getElementById("privacidad");
```

```
alert(" Elemento: " + elemento.value + "\n Seleccionado: " + elemento.checked);
```

- **Select:** para las listas desplegables se requiere obtener el valor (value) de la opción seleccionada (<option>). Para esto se utiliza:

.options → array para c/lista desplegable que contiene la referencia a todas las opciones de esa lista.

document.getElementById("id_de_la_lista").**option[0]** → obtiene la primera opción de la lista.

.selectedIndex → el índice de la opción seleccionada para el array options.

Ej:

```
<select id="opciones" name="opciones">
    <option value="1">Primer valor</option>
    <option value="2">Segundo valor</option>
    <option value="3">Tercer valor</option>
    <option value="4">Cuarto valor</option>
</select>
```

// Obtener la referencia a la lista

```
var lista = document.getElementById("opciones");
```

// Obtener el índice de la opción que se ha seleccionado

```
var indiceSeleccionado = lista.selectedIndex;
```

// Con el array "options", obtener la opción seleccionada

```
var opcionSeleccionada = lista.options[indiceSeleccionado];
```

// Obtener el valor y el texto de la opción seleccionada

```
var textoSeleccionado = opcionSeleccionada.text;
```

```
var valorSeleccionado = opcionSeleccionada.value;
```

```
alert("Opción seleccionada: " + textoSeleccionado + "\n Valor de la opción: " + valorSeleccionado);
```

-
- **Establecer el Foco en un Elemento:**

```
document.getElementById("primero").focus();
```

↓
<form id="formulario" action="#">

↑
<input type="text" id="primero" />
</form>

- **Evitar el Envío Duplicado de un Formulario:**

<form id="formulario" action="#">

...

<input type="button" value="Enviar" onclick="this.disabled=true;
this.value='Enviando...'; this.form.submit()" />

</form>

this.disabled=true → se deshabilita el botón.

this.form.submit() → se envía el formulario mediante la función submit().

| |
|---|
| Nota: solo sirve con type="button" |
|---|

- **Limitar el Tamaño de Caracteres en un Textarea:**

Algunos eventos como: onkeypress, onclick y onsubmit, se puede evitar su comportamiento normal si se devuelve el valor false.

<textarea onkeypress="return true;"> </textarea>

→ Se puede escribir cualquier carácter.


<textarea onkeypress="return false;"> </textarea>

→ No permite que ninguna tecla presionada se transforme en un carácter.

Aprovechando esta característica, se puede limitar el # de caracteres escritos en el textarea:

↓

```
function limita(maximoCaracteres) {  
    var elemento = document.getElementById("texto");  
    if(elemento.value.length >= maximoCaracteres ) {  
        return false;  
    }  
}
```



```

    else {
        return true;
    }
}

```

```

<textarea id="texto" onkeypress="return limita(100);"></textarea>


```

Este mismo ejemplo se puede mejorar indicando en todo momento al usuario el # de caracteres que aún puede escribir. Y permitiendo usar las teclas Backspace, Supr y las flechas horizontales cuando se ha llegado al máx. de caracteres.

```

var maximoCaracteres=100;

```



```

function limita(elEvento){
    var elemento=document.getElementById("texto");

    //obtener la tecla pulsada

    var evento=elEvento || window.event;
    var codigoCaracter=evento.charCode || evento.keyCode;

    //Permitir utilizar las teclas con flecha horizontal

    if(codigoCaracter == 37 || codigoCaracter == 39){
        return true;
    }

    //Permitir borrar con la flecha Backspace y con la tecla Supr

    if(codigoCaracter == 8 || codigoCaracter == 46){
        return true;
    }

    else if(elemento.value.length >= maximoCaracteres){
        return false;
    }
}

```

```

    }
    else{
        return true;
    }
}

function actualizaInfo(){
    var elemento=document.getElementById("texto");
    var info=document.getElementById("info");

    if(elemento.value.length >= maximoCaracteres){
        info.innerHTML="Se ha llegado al Máximo de "
        +maximoCaracteres+" caracteres";
        info.style.color='blue';
    }
    else{
        info.innerHTML="Puedes escribir hasta "+(maximoCaracteres-
        elemento.value.length)+" caracteres adicionales";
        info.style.color='green';
    }
}

window.onload=function(){
    var te=document.getElementById("texto");
    te.focus();
    te.onkeypress=limita;
    te.onkeyup=actualizaInfo;
}

<div id="info">Máximo 100 caracteres</div>

<textarea id="texto" rows="6" cols="30"></textarea>

```


Nota: este ejercicio corresponde al ejercicio 17, del capítulo 7.2 de JavaScript en Libros Web.

innerHTML → sirve para cambiar el texto HTML de la etiqueta.

Ej: <div>Este texto interno es el que cambia</div>

- **Restringir los Caracteres Permitidos en un Cuadro de Texto:**

```
function permite(elEvento, permitidos) {  
  
    // Variables que definen los caracteres permitidos  
  
    var numeros = "0123456789";  
  
    var          caracteres          = "  
    abcdefghijklmñopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";  
  
    var numeros_caracteres = numeros + caracteres;  
  
    var teclas_especiales = [8, 37, 39, 46];  
  
    // 8 = BackSpace, 46 = Supr, 37 = flecha izquierda, 39 = flecha derecha  
  
    // Seleccionar los caracteres a partir del parámetro de la función  
  
    switch(permitidos) {  
        case 'num':  
            permitidos = numeros;  
            break;  
        case 'car':  
            permitidos = caracteres;  
            break;  
        case 'num_car':  
            permitidos = numeros_caracteres;  
            break;  
    }  
}
```



// Obtener la tecla pulsada

```
var evento = elEvento || window.event;
```

```
var codigoCaracter = evento.charCode || evento.keyCode;
```

```
var caracter = String.fromCharCode(codigoCaracter);
```

// Comprobar si la tecla pulsada es alguna de las teclas especiales

// (teclas de borrado y flechas horizontales)

```
var tecla_especial = false;
```

```
for(var i in teclas_especiales) {
```

```
    if(codigoCaracter == teclas_especiales[i]) {
```

```
        tecla_especial = true;
```

```
        break;
```

```
    }
```

```
}
```

//Comprobar si la tecla pulsada se encuentra en los caracteres permitidos o si es una tecla especial

```
return permitidos.indexOf(caracter) != -1 || tecla_especial;
```

```
}
```

// Sólo números

```
<input type="text" id="texto" onkeypress="return permite(event, 'num')"/>
```

// Sólo letras

```
<input type="text" id="texto" onkeypress="return permite(event, 'car')"/>
```

// Sólo letras o números

```
<input type="text" id="texto" onkeypress="return permite(event, 'num_car')"/>
```

Al igual que el ejemplo del textarea, impide el comportamiento normal del evento onkeypress. Cuando se pulsa una tecla, se comprueba si el carácter está dentro de los permitidos.

Si se encuentra, devuelve true, y la tecla se escribe, sino, devuelve false y no deja escribir.

VALIDACIÓN DE FORMULARIOS:

- **Campo de Texto Obligatorio:** para que el usuario indique un valor en un cuadro de texto o textarea, de forma obligatoria:

```
valor = document.getElementById("campo").value;

if( valor == null || valor.length == 0 || /^s+$/ .test(valor) ) {
    return false;
}
```

/^s+\$/ .test(valor) → sirve para comprobar que el valor introducido, no esté formado solo por espacios en blanco.

Es una “expresión regular”.

- **Campo de Texto con Valores Numéricos:** se obliga a introducir un valor numérico.

```
valor = document.getElementById("campo").value;

if( isNaN(valor) ) {
    return false;
}
```

Resultados de la función isNaN():

isNaN(3); // false

isNaN("3"); // false

isNaN(3.3545); // false

isNaN(32323.345); // false

isNaN(+23.2); // false

```
isNaN("-23.2"); // false
```

```
isNaN("23a"); // true
```

```
isNaN("23.43.54"); // true
```

- **Seleccionar una opción de una lista:** se obliga al usuario a seleccionar un elemento de una lista desplegable.

```
indice = document.getElementById("opciones").selectedIndex;
```

```
if( indice == null || indice == 0 ) {  
    return false;  
}
```

```
<select id="opciones" name="opciones">  
    <option value="">- Seleccione un valor -</option>  
    <option value="1">Primer valor</option>  
    <option value="2">Segundo valor</option>  
    <option value="3">Tercer valor</option>  
</select>
```

La primera opción (- Seleccione un valor -) no es válida, por lo que no se permite el valor 0 para el índice.

- **Dirección de email válida:**

```
valor = document.getElementById("campo").value;
```

```
if( !( /\w+([-+.']\w+)*@\w+([-.\w+)*\.\w+([-.\w+)]/).test(valor) ) {  
    return false;  
}
```

- **Fecha válida:**

```
var anio = document.getElementById("anio").value;
```

```

var mes = document.getElementById("mes").value;

var dia = document.getElementById("dia").value;

valor = new Date(anio, mes, dia);

    if( !isNaN(valor) ) {
        return false;
    }

```

Date → es una function interna de JS, para contruir fechas. El # de mes se indica de 0 a 11 (0 → Enero, 11 → Diciembre).

En los días, el minimo permitido es 1 y el máx. es 31.

Las validaciones de los formularios, al no cumplir con las condiciones, devuelven false. Este valor previene el comportamiento normal del evento onsubmit.

```

<form action="" method="" id="" name="" onsubmit="return validación()"> ...
</form>

```

- **Validar que Todos los Chekbox han sido Seleccionados:**

Si se trata de comprobar que todos los checkbox del formulario han sido seleccionados, es más fácil utilizar un ciclo:

```

formulario = document.getElementById("formulario");

for(var i=0; i<formulario.elements.length; i++) {
    var elemento = formulario.elements[i];

    if(elemento.type == "checkbox") {
        if(!elemento.checked) {
            return false;
        }
    }
}

```

RELOJ EN PANTALLA:

Muestra la hora del pc del usuario.

```
function muestraReloj(){  
    var fechaHora=new Date();  
    var horas=fechaHora.getHours();  
    var minutos=fechaHora.getMinutes();  
    var segundos=fechaHora.getSeconds();  
    var sufijo="am";  
    if(horas>12){  
        horas=horas-12;  
        sufijo="pm";  
    }  
    if(horas<10){  
        horas="0"+horas;  
    }  
    if(minutos<10){  
        minutos="0"+minutos;  
    }  
    if(segundos<10){  
        segundos="0"+segundos;  
    }  
    document.getElementById("reloj").innerHTML=horas+":"+minutos+":"+  
    +segundos+sufijo;  
}
```

```
window.onload=function(){
```



```
    ↑
    |
    | setInterval(muestraReloj,1000);
    | }
    |
    | <div id="reloj"/>
```

Son métodos del objeto window:

setTimeout(nombreFuncion, milisegundos); → ejecuta una función después de que haya ocurrido el periodo de tiempo indicado. Solo se muestra 1 vez después de que se haya cargado la página.

setInterval() → igual a setTimeout, pero se ejecuta infinitas veces.

clearInterval(id_de_setInterval) → detiene.

Ej:

```
var intervalo=setInterval(algunaFuncion,3000);
// se detiene con:
clearInterval(intervalo);
```

MODELO DE EVENTOS DEFINIDOS POR W3C:

Implementa 1 método para todos los objetos. El método es:

addEventListener(evento, metodo_a_ejecutar,fase);

Le indica a un elemento html, que este pendiente de un evento.

Ej:

```
function resalta(elEvento){
    ...
}

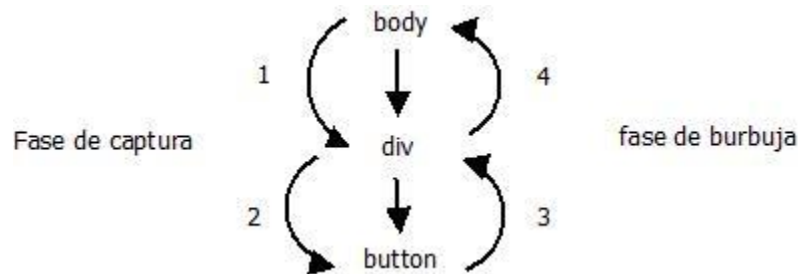
window.addEventListener('load',inicio);

function inicio(){
    document.getElementById("seccion").addEventListener('mouseover',
    resalta,false);
    document.getElementById("seccion").addEventListener('mouseout',
    resalta);
}
```

El parámetro fase, se puede obviar, casi siempre es false. Este parámetro se usa si existen funciones de escucha para elementos que contienen al elemento que se le genera el evento, es decir, al hacer clic en un div, también se le hace clic al body.

Al ser "false", salta la fase de captura y al ser "true", el flujo es normal.

Flujo de eventos:



.removeEventListener(); → quita la función de escucha de eventos.

Ej:

```
<body>
  <button id="miBoton">Haz clic </button>
</body>
```

```
window.addEventListener("load",function(){
    alert("La pág. se cargó");
});

var boton=document.getElementById("miBoton");
boton.addEventListener("click",boton_click);

function boton_click(){
    alert("Bienvenido");
    boton.removeEventListener("click",boton_click);
}
```


OBJETOS:

Nota: ver Capitulo 3 – AJAX –Libros Web.

Para hacer programación orientada a objetos, se puede usar:

```
var ObjetoEjemplo=new claseEjemplo();
```

//constructor con atributos y métodos:

```
function claseEjemplo(){  
|  
}
```

Ej:

```
var Gato=new gato("Pelusa");
```

```
var Gato2=new gato("Mishka");
```

```
function gato(nombre){  
|  
    //atributo  
    this.nombre=nombre;  
|  
    //método  
    this.maulla=function(){  
|  
        alert("miaww");  
|  
    }  
|  
}
```

```
alert("El gato se llama "+Gato.nombre);
```

```
alert("El gato es "+Gato2.nombre);
```

//llamada al método:

```
Gato.maulla();
```

| |
|--|
| Nota: en JavaScript un objeto es igual a un array asociativo. |
|--|

NOTACIÓN JSON:

(JavaScript Object Notation)

Formato para el intercambio de info.

Permite representar estructuras de datos (arrays) y objetos (arrays asociativos) en forma de texto.

Ej:

- Notación Array Tradicional:

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

```
modulos[0] = "Lector RSS";
```

```
modulos[1] = "Gestor email";
```

```
modulos[2] = "Agenda";
```

```
modulos[3] = "Buscador";
```

```
modulos[4] = "Enlaces";
```

- **Notación Array JSON:**

```
var modulos=["Lector RSS", "Gestor email", "Agenda", "Buscador", "Enlaces"];
```

- Notación Array Asociativo:

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

```
modulos.titulos = new Array();
```

```
modulos.titulos['rss'] = "Lector RSS";
```

```
modulos.titulos['email'] = "Gestor de email";
```

```
modulos.titulos['agenda'] = "Agenda";
```

- **Notacion Array Asociativo JSON:**

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

```
modulos.titulos = {rss: "Lector RSS", email: "Gestor de email", agenda: "Agenda"};
```

Notación JSON para Arrays Asociativos:

- 1) Los contenidos del array asociativo se encierra entre llaves { }
- 2) Los elementos del array se separan mediante coma ,
- 3) La clave y el valor de c/elemento se separan mediante dos puntos :

Si la clave no tiene espacios en blanco se puede prescindir de las comillas dobles, pero si tiene espacios en blanco, son obligatorias.

```
var titulos={"Lector RSS": "rss"};
```


Ejercicio 1: (de AJAX – Libros Web)

Definir la estructura de un objeto que almacena una factura. Las facturas están formadas por la información de la propia empresa (nombre de la empresa, dirección, teléfono, NIF), la información del cliente (similar a la de la empresa), una lista de elementos (cada uno de los cuales dispone de descripción, precio, cantidad) y otra información básica de la factura (importe total, tipo de iva, forma de pago).

Una vez definidas las propiedades del objeto, añadir un método que calcule el importe total de la factura y actualice el valor de la propiedad correspondiente. Por último, añadir otro método que muestre por pantalla el importe total de la factura.

// Estructura básica del objeto Factura

```
var factura = {  
    empresa: {  
        nombre: "Nombre de la empresa",  
        direccion: "Dirección de la empresa",  
        telefono: "77777777",  
        nif: ""  
    },  
}
```





```
cliente: {  
    nombre: "Nombre del cliente",  
    direccion: "Dirección del cliente",  
    telefono: "77777770",  
    nif: ""  
},
```

```
elementos: [  
    {  
        descripcion: "Producto 1",  
        cantidad: 0,  
        precio: 0  
    },  
    {  
        descripcion: "Producto 2",  
        cantidad: 0,  
        precio: 0  
    },  
    {  
        descripcion: "Producto 3",  
        cantidad: 0,  
        precio: 0  
    }  
],
```

```
informacion: {  
    baseImponible: 0,  
    iva: 16,  
    total: 0,  
    formaPago: "contado"  
}
```

```
};
```

// Métodos de cálculo del total y de visualización del total:

```
factura.calculaTotal = function() {  
    for(var i=0; i<this.elementos.length; i++) {  
        this.informacion.baseImponible += this.elementos[i].cantidad *  
        this.elementos[i].precio;  
    }  
  
    var importelva = (1 + (this.informacion.iva/100));  
  
    this.informacion.total = (this.informacion.baseImponible *  
    importelva).toFixed(2);  
}
```

```
factura.muestraTotal = function() {  
    this.calculaTotal();  
  
    alert("TOTAL = " + this.informacion.total + " euros");  
}
```

//llamado método:


```
factura.muestraTotal();
```

Ejercicio Factura con Pseudoclases:

// Definición Clase Cliente:

```
function Cliente(nombre, direccion, telefono, nif) {  
    this.nombre = nombre;  
    this.direccion = direccion;  
    this.telefono = telefono;
```





```
    this.nif = nif;
}
```

// Definición Clase Elemento:

```
function Elemento(descripcion, cantidad, precio) {

    this.descripcion = descripcion;

    this.cantidad = cantidad;

    this.precio = precio;

}
```

// Definición Clase Factura

```
function Factura(cliente, elementos) {

    this.cliente = cliente;

    this.elementos = elementos;

    this.informacion = {

        baseImponible: 0,

        iva:          16,

        total:        0,

        formaPago:    "contado"

    };

};
```


// La información de la empresa que emite la factura se añade al prototype, porque se supone que no cambia

```
Factura.prototype.empresa = {

    nombre:  "Nombre de la empresa",

    direccion: "Direccion de la empresa",

}
```



```
↑
    telefono: "900900900",
    nif:      "XXXXXXXXX"
};
```

// Métodos añadidos al prototype de la Factura

```
Factura.prototype.calculaTotal = function() {
    for(var i=0; i<this.elementos.length; i++) {
        this.informacion.baseImponible += this.elementos[i].cantidad *
        this.elementos[i].precio;
    }

    this.informacion.total = this.informacion.baseImponible *
    this.informacion.iva;
}
```

```
Factura.prototype.muestraTotal = function() {
    this.calculaTotal();
    alert("TOTAL = " + this.informacion.total + " euros");
}
```

// Creación de una factura

```
var elCliente = new Cliente("Cliente 1", "", "", "");

var losElementos = [new Elemento("elemento1", "1", "5"),
    new Elemento("elemento2", "2", "12"),
    new Elemento("elemento3", "3", "42")
];

var laFactura = new Factura(elCliente, losElementos);

laFactura.muestraTotal();
```

CICLO FOR IN:

```
var mensaje="";
var variables="";
var persona={
    nombre : "Juan",
    apellido : "Perez",
    edad : 7
};
for(var i in persona){
    mensaje=mensaje+persona[i];
    variables=variables+i;
}
alert(mensaje);
//imprime JuanPerez7
alert(variables);
//imprime nombreakellidoedad
```

MÉTODOS APPLY() Y CALL():

Útiles para las funciones, ambos permiten ejecutar una función como si fuera un método de otro objeto.

Ej: se usa **call()** para ejecutar una función como si fuera un metodo del objeto: "elObjeto".

```
function miFuncion(x){
    return this.numero+x;
}

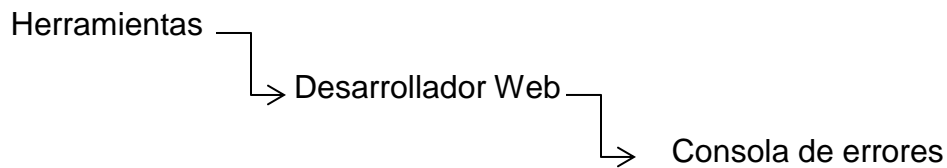
var elObjeto=new Object();
elObjeto.numero=5;
var resultado=miFuncion.call(elObjeto,4);
alert(resultado); //imprime 9
```


- ➔ Acá “this”, dentro de la función se refiere al objeto enviado en call()
- ➔ Los demás parámetros (4), se refieren a los parámetros de la función.

.apply() ➔ es igual al método call(), solo que los parámetros se pasan como un array.

```
var resultado=miFuncion.apply(elObjeto,[4]);
```

VER ERRORES DE CÓDIGO JAVASCRIPT EN FIREFOX:



O utilizar el complemento “FireBug”

ACTUALIZACIÓN ACCESO A NODOS:

Podemos acceder a un id, o una clase, o a cualquier elemento usando selectores css, como en jquery de la siguiente forma:

```
record = document.querySelector('record')
```

En este caso accederíamos a un elemento con id=”record”.

RECURSOS ÚTILES:

- **Light Box:** herramienta muy útil para mostrar galerías de imágenes.

<http://huddletogogether.com/projects/lightbox2/>

<http://lokeshdhakar.com/>

<http://lokeshdhakar.com/projects/lightbox2/>

- **Color Thief:** herramienta que nos indica cual es el color predominante en una imagen, y toda la paleta de colores que usa.

<http://lokeshdhakar.com/projects/color-thief/>

- **Hotscripts:** sitio web en inglés que contiene referencias a miles de scripts y utilidades de JavaScript gratuitas.

<http://www.hotscripts.com/category/scripts/javascript/>

- **Dynamic Drive:** sitio web en inglés con cientos de scripts gratuitos de JavaScript con las utilidades más comunes (calendarios, menús, formularios, animaciones de texto, enlaces, galerías de imágenes, etc.)

<http://www.dynamicdrive.com/>

- **Gamarod JavaScript:** sitio web en español que contiene decenas de utilidades JavaScript gratuitas listas para usar.

<http://www.gamarod.com.ar/>

- **Expresiones Regulares:** para crear y entender las expresiones regulares.

<https://regex101.com/>

