http://www.w3schools.com/js/default.asp

NOCIONES BÁSICAS

Introducción a JavaScript

JavaScript es un lenguaje de programación interpretado que se utiliza fundamentalmente para dotar de **comportamiento dinámico** a las páginas web. Fue diseñado para que su código se incrustase en documentos HTML y fuera interpretado por el navegador web.

Mediante JavaScript nos encargaremos de realizar acciones en el cliente, como pueden ser pedir datos, confirmaciones, validación de datos en formularios, incorporar efectos como texto que aparece y desaparece, animaciones, acciones que se activan al pulsar botones, manejar ventanas con mensajes de aviso al usuario….

Cualquier **navegador** web actual incorpora un intérprete para código JavaScript.

JavaScript ha sido elegido por la W3C (World Wide Web Consortium), consorcio responsable de la especificación del lenguaje HTML, como estándar para HTML5.

Actualmente la responsable del desarrollo de JavaScript es la Mozilla Foundation

Dónde y cómo incluir JavaScript

Existen dos modos de incluir lenguaje JavaScript en una página:

- A través del elemento **<script>:**

  El código JavaScript se encierra entre etiquetas <script> </script> que marcan el principio y fin del bloque de código JavaScript de modo que el navegador sabe que lo contenido entre estas etiquetas no corresponde a código HTML, si no que se trata de código que debe ser procesado antes de mostrar el resultado en pantalla.

  El formato es el siguiente:

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

Actualmente se puede omitir el atributo type, ya que todos los script están codificados en javascript.

A su vez, el elemento <script> nos permite aplicar dos técnicas diferentes para insertar el código:

- Escribir el código JavaScript directamente en el documento HTML (esta técnica se denomina incrustación)

Es correcto incluir cualquier bloque de código en cualquier zona de la página, en el body o en el head.

```
<html>
<head>
<title> PROBANDO JAVASCRIPT </title>
</head>
<body>
…
<script type="text/javascript">
alert ("bienvenido");
</script>
</body>

</html>
```

- Hacer referencia a un archivo independiente que contiene el código JavaScript (esta técnica se denomina inclusión).

```
<html>
<head>
<title> PROBANDO JAVASCRIPT </title>
</head>
<body>
…
<script  type="text/javascript" src="saludo.js"  > </script>

</body>

</html>
```

saludo.js contendrá el código:

```
alert ("bienvenido");
```

El archivo tendrá extensión js

Es la forma más adecuada de trabajar pues al localizar el código JavaScript en un fichero externo facilitamos la reutilización y modularización de dicho código

y aprovechar las funcionalidades de caché de los navegadores (los navegadores almacenan una copia local de los archivos de código JavaScript y, si otra página visitada por el usuario requiere ese mismo archivo, utilizan la versión local en lugar de volver a descargarla de nuevo)

*Se recomienda colocar los elementos <script> lo más abajo posible dentro del body. En **HTML5** se ha añadido el atributo **async** al elemento script, que sirve para indicar al navegador que no es necesario que detenga la carga de la página hasta que se haya completado la descarga del archivo js (todavía no es compatible en todos los navegadores).*

- Otra manera de incluir código JavaScript en directamente **como respuesta a algún evento**:

```
<html>
<head>
</head>
<body>
<input type="submit"
  onclick="alert('Acabas de hacer click');return false;" value="Click">
</body>
</html>
```

Este uso lo veremos más adelante.

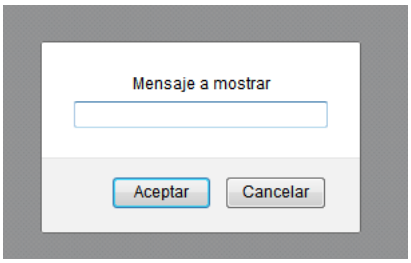
Normas sintácticas

- Sensible a mayúsculas/minúsculas
  - Sintaxis lowerCamelCase si están compuestas por varias palabras
- Comentarios
  - //
  - /* */
- Fin de la instrucción con ;
  Podemos omitir dicho signo si cada instrucción se encuentra en una línea independiente pero no es una buena práctica de programación.
- Espacios consecutivos se consideran un único espacio

Leer datos de teclado

```
var n1 = prompt ('Mensaje a mostrar');
```

Muestra:



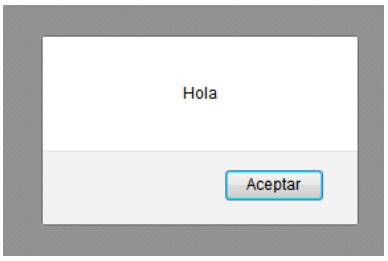
El dato anotado siempre se recoge como una cadena. Si se trata de un número deberemos convertirle a ese tipo usando la función adecuada (más adelante).

Escribir un dato

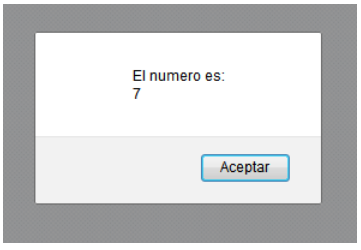
Hay dos formas básicas:

alert

```
alert('Hola');
```




```
num=7;
alert('El numero es\n:'+num);
```



document.write : Sin alerta. No admite secuencias de escape, sí código html.

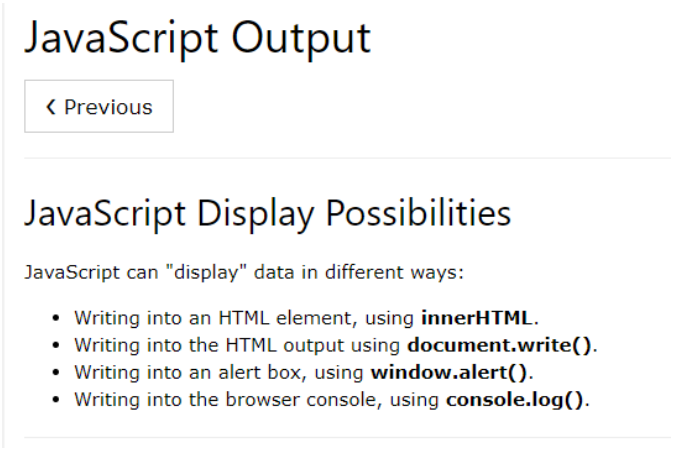
```
num=7;
document.write('El numero es:<br>'+num);
```

El numero es:

7

Ver

http://www.w3schools.com/js/js_output.asp



JavaScript Output

‹ Previous

JavaScript Display Possibilities

JavaScript can "display" data in different ways:

- Writing into an HTML element, using **innerHTML**.
- Writing into the HTML output using **document.write()**.
- Writing into an alert box, using **window.alert()**.
- Writing into the browser console, using **console.log()**.

Variables y tipos de datos

- Permite declarar variables mediante **var**

```
var a=7, b=’pepe’;
```

  pero no obliga a hacerlo.
- Podemos inicializar una variable sin utilizar var, de modo que JavaScript declara implícitamente esa variable.
- No se puede utilizar una variable que no haya sido declarada o al menos inicializada.

- Lenguaje de programación de "tipado débil una misma variable puede contener datos de tipo diferente en distintos instantes.

```
a=25;
alert (typeof a ); // number
a= 'hola';
alert (typeof a ); // string
```

- Una variable declarada con var dentro de una función tiene alcance local es decir, será reconocida dentro de esa función pero no fuera de la función.
- Una variable declarada con var fuera de una función tiene alcance global.
- Si existe una variable local en una función con el mismo nombre que una variable global, dentro de la función se asocia ese nombre a la variable local.
- Cualquier variable usada sin haber sido declarada tiene alcance global (puede ser utilizadas en el interior de cualquier función o en cualquier otro código incluido en la misma página web)
- Identificadores de variables:

  Se pueden utilizar caracteres alfanuméricos y el carácter subrayado _

  No pueden comenzar por número.

  No pueden utilizarse palabras reservadas.

  Palabras reservadas: http://msdn.microsoft.com/es-es/library/0779sbks(v=vs.94).aspx

break	delete	if	this	while
case	do	in	throw	with
catch	else	instanceof	try	
continue	finally	new	typeof	
debugger	for	return	var	
default	function	switch	void	

Tipos de datos en Javascript

Los valores que se pueden almacenar en una variable se corresponden con los siguientes tipos:

- **number**

  Cualquier número entero o decimal (expresado en notación decimal, hexadecimal o científica)

  var n1=4500;

  var n2=4500.00;

  var n3=0x1194; // notación hexadecimal

  var n4=4.5e3; // notación científica

- **string**

  Cualquier cadena de caracteres comprendida entre comillas dobles o simples

  var texto1 = "Hola caracola";

  var texto2 = 'Buenas…';

No obstante, a veces las cadenas de texto contienen tanto comillas simples como dobles. Además, existen otros caracteres que son difíciles de incluir en una variable de texto (tabulador, ENTER, etc.) Para resolver estos problemas, JavaScript define un mecanismo para incluir de forma sencilla caracteres especiales y problemáticos dentro de una cadena de texto.

El mecanismo consiste en sustituir el carácter problemático por una combinación simple de caracteres. A continuación se muestra la tabla de conversión que se debe utilizar:

Si se quiere incluir...	Se debe incluir...
Una nueva línea	\n
Un tabulador	\t
Una comilla simple	\'
Una comilla doble	\"
Una barra inclinada	\\

Ejemplo:

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

Este mecanismo de JavaScript se denomina "mecanismo de escape" de los caracteres problemáticos, y es habitual referirse a que los caracteres han sido "escapados".

También podemos incluir cualquier carácter Unicode de 16 bits utilizando la fórmula \u*nnnn*, donde *nnnn* es el valor hexadecimal del carácter. En http://unicode.org/charts/ pueden consultarse todos los caracteres Unicode, como los símbolos matemáticos, los caracteres de otros idiomas o los signos Braille. En la siguiente tabla se recogen los códigos Unicode de algunos caracteres especiales del castellano, y en la dirección http://www.rishida.net/tools/conversion/ dispone de un completo conversor:

á \u00e1	é \u00e9	í \u00ed	ó \u00f3	ú \u00fa	Á \u00c1	É \u00c9	Í \u00cd
Ó \u00d3	Ú \u00da	ü \u00fc	Ü \u00dc	ñ \u00f1	Ñ \u00d1	¡ \u00a1	¿ \u00bf

- **boolean**

  Admite los datos false y true

- **undefined**

  Valor asignado implícitamente a todas las variables declaradas a las que no se les ha asignado dato aun.

- **null**

Se utiliza para inicializar una variable que posteriormente vamos a referir a un objeto pero aun no tiene objeto asignado.

```
var automovil=null;
```

Si utilizamos typeof para ver el tipo de automóvil nos dirá que es un object.

null es un convenio que se ha implementado como un objeto vacío.

También se utiliza null para preguntar si una variable no tiene valor.

```
var texto;
if (texto == null)
        alert ('texto no tiene valor asignado');
```

El valor **null** se comporta como el número 0, mientras que **undefined** se comporta como el valor especial **NaN** (no es un número). Si comparas un valor **null** con un valor **undefined**, son iguales.

```
var a;
b=null;
alert (a); //undefined
alert (b); // null
alert (a+2); // NaN
alert (b+2);  // b lo considera como 0 y da 2
alert (a==b);  // true
alert (a==undefined); // true
alert (a==null); // true
```

OPERADORES

Operadores aritméticos

Operador	Nombre	Descripción
++	Incremento unario	Suma 1 a una variable. Puede usarse en modo prefijo ++a o sufijo a++;
--	Decremento unitario	Resta 1 a una variable.Pude usarse en modo prefijo--a o sufijo a--;
+	Suma	Por ejemplo a = 5 + 3.2; //a = 8.2
-	Resta	Por ejemplo a = 5 - 3.2; //a = 1.8
*	Producto	Por ejemplo, a = 5 * 3.2; // a = 16
/	Cociente	Por ejemplo, a = 5 / 3.2; //a = 1.5625
%	Resto o módulo	Por ejemplo, a = 5 % 3; //a = 2

Operadores booleanos o lógicos

Operador	Nombre	Descripción
!	Negación	Por ejemplo, a = !false; // a = true
&&	Y	Por ejemplo, a = true && false; //a = false
||	O	Por ejemplo a= true || false; //a = true

Nota: En JavaScript los operadores booleanos && y || son de tipo cortocircuito lógico. ¿Qué quiere decir esto? Que si el operando de la izquierda es suficiente para conocer el resultado de la operación, no se evalúa el de la derecha .

Operadores de comparación

Operador	Nombre	Descripción

==	Igual que	Por ejemplo, 5 == 3 + 2; //true
		Pero 0.3 == 0.1 + 0.2; //false
		Y false == "false"; //false
		Pero false == "0"; //true
		Y "2" == 2; //true

!=	Distinto que	Por ejemplo, 5 != 3 + 2// false
===	Idéntico a (incluso en tipo)	Por ejemplo, "2" === 2; //false
!==	Diferente a (en dato y/o tipo)	Por ejemplo, "2" !== 2;//true

<	Menor que (números), o código ASCII anterior a (cadenas)	Por ejemplo, 5 < 3; //false
		Y "casa" < "tos"; //true
		Pero "Tos" < "casa"; //true

<=	Menor o igual que, o código anterior o igual que	Por ejemplo, 5 <= 3; //false
>	Mayor que (números), o código ASCII posterior a (cadenas)	Por ejemplo, 5 > 3; //true
>=	Mayor o igual que, o código posterior o igual que	Por ejemplo, 5 >= 3; //true

Operadores de asignación

Operador	Nombre	Descripción
=	Asignación simple	Por ejemplo, nombre = 'pepe';

+=	Suma/concatenación y asignación	Por ejemplo, nombre += apellido;
-=	Resta y asignación	Por ejemplo, credito-=1;
*=	Producto y asignación	Por ejemplo, iva*=1.1;
/=	Cociente y asignación	Por ejemplo, salario /= 1.1;
%=	Resto y asignación	Por ejemplo, dni =% 23;

Otros operadores de interés

Operador	Nombre	Descripción
?:	Operador condicional	Si la expresión anterior al signo ? se evalúa como true el resultado es el dato anterior al signo :, y en caso contrario el dato posterior al signo :. Por ejemplo, temperatura>36.5?'enfermo':'sano'
typeof	Tipo de	Calcula el tipo del dato. Por ejemplo, typeof 'casa' alert(typeof 'casa'); → string alert(typeof 23); → number alert(typeof null); → object
+	Concatenación de cadenas	Sirve para concatenar dos cadenas de caracteres.
( )	Agrupación de operaciones	Se utiliza para agrupar operaciones estableciendo en qué orden deben efectuarse.

Precedencia de los operadores

()
++ -- !
* / %
+ -
< > <= >=
== !=
&&
| |
= += -= *= /= %=

Conversión de tipos implícita

- true se convierte en 1 antes de intervenir en una operación de comparación.
  alert ((5==2+3)==2); →false
  alert ((5==2+3)==1); →true

- false se convierte en 0 antes de intervenir en una operación de comparación
  alert ((5==2+2)==0); → true

- Al comparar una cadena con un número se convertirá la cadena en un número antes.
  alert (23=="23"); → true

- En una comparación, cualquier cadena no vacía que pueda leerse como un número expresado en decimal (incluida notación científica) o hexadecimal se convertirá en ese número.

- Cuando en una operación con el operador + uno de los operandos es una cadena, el otro se convertirá también en una cadena.
  alert (2+2); → 4
  alert (2+'2'); → 22

Se recomienda:

Evitar mezclar tipos de datos en las operaciones cuando sea posible

Recurrir a alguno de los métodos de conversión explícito o funciones de forzado de tipo (*casting*) que se recogen en la siguiente tabla cuando no lo sea.

Función/Método	Descripción
Number(*dato*)	Función que devuelve el resultado de convertir *dato* en un número (que puede ser NaN Not a Number). true se convierte en 1, y false se convierte en 0. Cualquier cadena que pueda ser interpretada como un número ('12.45', '0x3f', '314e-2') se convierte en ese número. Una cadena vacía se convierte en 0. Cualquier otro dato se convierte en NaN. La función **isNaN**(valor) devuelve true cuando valor vale NaN y false en caso contrario

```
a='23';
alert (a+5); // 235
```

```
a=Number (a);
alert (a+5);  // 28
alert (Number("1.23")+4); // 5.23
alert(Number('casa')); // NaN
alert(Number('')); // 0
alert(Number(true)); // 1
alert(Number(false)); // 0
```

parseInt(*cadena*) Función que analiza *cadena* carácter a carácter empezando por la izquierda intentando componer un número entero expresado en decimal o hexadecimal. En cuanto encuentra un carácter que no puede formar parte de un número entero detiene el análisis y devuelve el resultado del análisis hasta ese punto. Si el análisis se detiene en el primer carácter o *cadena* está vacía devuelve NaN (obsérvese que este comportamiento es diferente al del Number(), que devuelve 0).

```
a='23';
alert (a+5); // 235
a=parseInt (a);
alert (a+5);  // 28
alert (parseInt("1.23")+4); // 5
alert (parseInt("1.83")+4); // 5
alert (parseInt("1pa83")+4); // 5
alert(parseInt('casa')); // NaN
alert(parseInt('')); // NaN
alert(parseInt(true)); // NaN
alert(parseInt(false)); // NaN
```

parseFloat(*cadena*) Similar a la función anterior, pero intentando componer un número de coma flotante expresado en decimal o notación científica.

```
alert (parseFloat ("1.23")+4); // 5.23
alert (parseFloat ("1.83")+4); // 5.83
alert (parseFloat ("1,83")+4); // 5
alert (parseFloat ("1pa83")+4); // 5
```

Boolean(*dato*) Función que devuelve el resultado de convertir *dato* a un valor booleano (true o false).

Cualquier cadena no vacía y cualquier número distinto de 0 se

convierte en true, incluso "false". Una cadena vacía y el número 0 se convierten en false.

```
alert (Boolean(1)); // true
alert (Boolean(0)); // false
alert (Boolean(32));  //true
alert (Boolean('casa'));  //true
alert (Boolean(''));  // false
alert (Boolean(true));    //true
alert (Boolean(false));   // false
alert (Boolean('true'));  //true
alert (Boolean('false')); //true
```

String(*dato*) Función que devuelve el resultado de convertir *dato* en una cadena de caracteres.

```
alert (23.567+5); // 28.567
alert (String(23.567)+5);  // 23.5675
```

BIFURCACIONES

- **If**

La sintaxis básica de la instrucción if es la siguiente:

if (*expresiónDeControl*){

instrucciones a ejecutar si la expresión de control es verdadera;

}else{

instrucciones a ejecutar si la expresión de control es falsa;

}

```
var a = 5, b= 3;
if (a<b) {
        alert (b+' es mayor que '+a);}
else
        if (a==b) {
                alert ('son iguales');}
        else
                alert (b+' es menor que '+a);
```

- **switch case**

Las bifurcaciones de tipo switch…case comparan el valor de una expresión de control con diferentes "casos" posibles, y ejecutan el código asociado al caso correspondiente, o el bloque correspondiente al caso default (por defecto). Es muy importante recordar que cada bloque deberá terminar con una instrucción break si no queremos que se ejecuten automáticamente las instrucciones del siguiente caso (aunque su valor no coincida con el de la expresión de control), y así sucesivamente hasta que se encuentre un break o se alcance el final de la instrucción switch…case.

Su sintaxis es la siguiente:

switch (*expresiónDeControl*){

case *valor1*:
bloque de instrucciones a ejecutar si expresiónDeControl coincide con valor1;
break;
case *valor2*:
b*loque de instrucciones a ejecutar si expresiónDeControl coincide con valor2;*
break;
…
default:
bloque de instrucciones a ejecutar si expresiónDeControl no coincide con ningún caso;
}

```
switch (a){
        case 1:
                alert ('a vale 1');
                break;
        case 2:
                alert ('a vale 2');
                break;
        case 5:
                alert ('a vale 5');
                break;
        default:
                alert ('no vale nada');
}
```

BUCLES

- **while**

La sintaxis de while es la siguiente:

while (*expresiónDeControl*){

bloque de instrucciones a ejecutar en cada iteración;

}

```
var a=1;
while (a<=5){
        document.write ("<br> a vale "+a);
        a++;
}
```

- **do while**

La sintaxis de do … while es la siguiente:

do{

bloque de instrucciones a ejecutar en cada iteración;

}while (*expresiónDeControl*);

```
var a=1;
do{
        document.write ("<br> a vale "+a);
        a++;
}
while (a<5);
```

- **for**

La sintaxis de for es la siguiente:

for(*instrucciónDeInicialización*;*expresiónDeControl*;*instrucciónTrasCadaIteración*){

bloque de instrucciones a ejecutar en cada iteración;

}

```
for (a=1;a<=5;a++)
        document.write ("<br> a vale "+a);
```

break → Para forzar la salida de un bucle

continue → Para abandonar la iteración sin ejecutar las instrucciones restantes y evaluar de nuevo la expresión de control para decidir si debe realizar otra iteración o no