JavaScript

manual de referencia

Autor: Jorge Sánchez (www.jorgesanchez.net) año 2003

Basado en el lenguaje JavaScript compatible para los navegadores Explorer y Netscape.

Versión normalizada de la ECMA (llamado ECMAScript) en:
http://www.ecma-international.org/publications/standards/Ecma-262.htm

nociones básicas

Java y Javascript

Java es un lenguaje de programación (como el Pascal, el BASIC o el C y C++) que fue desarrollado por la empresa **Sun** fundamentalmente para crear aplicaciones para Internet. El lenguaje Java es completo, es decir permite realizar cualquier operación sobre el ordenador (como por ejemplo borrar un archivo) y su aprendizaje es costoso.

Javascript es lo que se conoce como **lenguaje script**, es decir: se trata de código de programación que se inserta dentro de un documento. Javascript fue desarrollado por la empresa **Netscape** con la idea de potenciar la creación de páginas Web dinámicas para su navegador **Navigator.**

Javascript (en contra de lo que se podría suponer) es totalmente distinto de Java. Java crea programas totalmente independientes y operativos; Javascript es más sencillo porque lo único que permite es insertar código especial dentro del HTML de una página, su función es ampliar las posibilidades de HTML. Javascript no crea programas independientes, dependen por completo del código HTML de la página.

El código en Java se debe compilar (convertir en instrucciones del ordenador) y entonces podrá ser utilizado por los navegadores (son las famosas **applets**). Sin embargo Javascript es interpretado directamente por el navegador; de hecho el código Javascript se incrusta dentro del código HTML de la página.

Java no puede acceder a los elementos HTML de una página (ya que su funcionalidad es mucho mayor) sin embargo Javascript necesita acceder a ellos, de otro modo no tendría sentido su uso.

La ventaja fundamental de Javascript es que su aprendizaje y uso son muy sencillos y que permite realizar labores complejas en una página sin necesidad de aprender CGI.

versiones de Javascript

Puesto que JavaScript fue desarrollado por Netscape, los navegadores de esta empresa lo incluyen desde la versión 2. Microsoft por su parte incluyó en la versión 3 una variante de este código llamado **JScript** que es casi idéntico al original JavaScript. Después se estandarizó el lenguaje, aunque ambas compañías poseen elementos que no son comunes con el estándar (aunque prácticamente todo el estándar es reconocido por ambas).

Así aparición el JavaScript 1.1 que es admitido por Navigator 3 y por Explorer 4. Y las versiones 1.2 y 1.3 que son reconocidas por las versiones 4 y posteriores de ambos navegadores.

La ECMA (asociación industrial para la normalización) definió un lenguaje estándar llamado **ECMAScript** que intentaba agrupar a los anteriores e incluía instrucciones nuevas

http://www.ecma-international.org/publications/standards/Ecma-262.htm

inclusión de Javascript en las páginas

Para hacer que un documento HTML incluya instrucciones en Javascript se debe hacer uso de la etiqueta **<SCRIPT>** de esta forma:

```
<script language="JavaScript">
   código JavaScript
</script>
```

Si se quiere especificar qué versión de Javascript se utiliza, para evitar que navegadores que no soportan la versión decodifiquen el Javascript, entonces se usa, por ejemplo:

```
<script language="JavaScript1.3">
```

navegadores no compatibles

Los navegadores que no soportan Javascript, no interpretarían las instrucciones Javascript sino que mostrarían el texto de las instrucciones en la página. Para evitar que estos navegadores lean el código en Javascript se hace:

```
<script language="Javascript">
<!-
   código JavaScript
//-->
</script>
```

El signo "<!-" indica principio de comentario en HTML y el signo "-->" indica fin de comentario. A su vez el signo "//" indica comentario en Javascript (el intérprete de Javascript no tendrán en cuenta esa línea).

uso de un archivo externo

También se puede utilizar el código JavaScript escrito en un archivo separado. Este archivo debe tener la extensión **js.** En el archivo se coloca sólo código en JavaScript. Después ese código se puede invocar desde la página web con el código:

```
<script language="Javascript" src="archivo.js">
```

normas de escritura en Javascript

- Los comentarios deben empezar con el símbolo // si son de una sola línea o iniciarse con los símbolos /* y finalizar con */ si son de varias líneas.
- Las líneas de código terminan con el signo de punto y coma (**;**)
- Javascript distingue entre mayúsculas y minúsculas
- Las llaves ({ y }) permiten agrupar código.

variables y operadores

variables

Una variable es un elemento que tiene un determinado nombre y que permite almacenar valores.

nombre de las variables

Deben empezar con una letra la cual puede ir seguida de números, el signo "_" o más letras.

valores

Los valores que pueden asignarse a una variable pueden ser:

- **Cadenas de texto**: "esto es una prueba", 'prueba' o "esto es una 'prueba' de código". Siempre se encierran entre comillas dobles o simples. Una variable de texto que no tiene contenido, se dice que tiene valor **null**. La palabra **null** es un término reconocido por Javascript.
- **Valores numéricos**: 1, -100, 1.6, 2.0E2.
- **Valores booleanos: true o false.**

caracteres especiales

Los valores de tipo texto van entre comillas y dentro de ellos se pueden colocar caracteres especiales (caracteres que no se pueden ver, como el cambio de línea) los cuales son:

- **\a**: Alarma
- **\b**: Retroceso (cursor una posición hacia atrás).
- **\f**: Nueva página de impresora
- **\n**: Nueva línea
- **\r**: Retorno de carro
- **\t**: Tabulador
- \\: Signo "\"

declaración de una variable

Para declarar una variable se puede emplear:

```
var variable = valor;
```

O simplemente:

```
variable = valor;
```

De tal modo, que realmente en Javascript no hace falta declarar una variable antes de su uso. Ejemplos:

```
var testear = 0;
testeaTexto = "Mi casa";
SeleccionarColor = true;
```

JavaScript permite que una variable pueda almacenar distintos tipos de datos en cada trozo de código. Es decir, una variable que ahora almacena texto, después puede almacenar números.

Tras declarar la variable, su valor puede cambiar mediante la asignación de un valor:

```
Testear = 12.3;
```

O mediante la asignación del resultado de una operación:

```
Testear = 12 * 3 + varX;
```

conversión de datos

En muchas lenguajes si una variable toma valores de texto y luego se quiere hacer que tome números, resulta imposible hacerlo. No es el caso de JavaScript ya que realiza conversiones implícitas. Ejemplo:

```
var x="50"   //x es una variable de texto
var y=30     //y es una variable numérica
z1=x+y       //z1 es variable de texto y vale "5010"
z2=y+x       //z2 es numérica y vale 60
/*dependiendo de cuál sea el primer operando, se determina
el tipo del resultado*/
```

Naturalmente ocurrirá un error si pretende convertir a un número, un texto normal como "Hola" por ejemplo. En cualquier caso no conviene hacer conversiones de tipo en ningún caso.

operadores

Los operadores son los elementos que permiten realizar operaciones con los datos del código.

operadores aritméticos

Operador	Significado
+	Suma
-	Resta
*	Multiplicación
/	Dividir
%	Resto de la división
++	Incremento
--	Decremento

Ejemplo:

```
var valor1=50;
var valor2=10;
var valor3=20;
var suma, resta, producto, division, resto;
var incremento, decremento;

suma=valor1+valor2; //suma vale 60
resta=valor1-valor2;  //resta vale 40
producto=valor1*valor2;  //producto vale 5000
division=valor1/valor3;  //division vale 2,5
resto=valor1%valor3; //resto vale 10

valor1++; //valor1 vale 51
valor1--; //valor1 vale 50

incremento=valor1++; //incremento vale 50 y valor1 vale 51
decremento=valor1--; //decremento vale 51, valor1 vale 50

incremento=++valor1; //incremento vale 51 y valor1 también
decremento=--valor1; //decremento y valor1 valen 50
```

operadores lógicos

Trabajan con proposiciones matemáticas (valores boléanos) son:

Operador	Significado
&&	AND (Y lógico)
||	OR (O lógico)
!	NOT (NO lógico)

operadores de comparación

Son:

Operador	Significado
==	Igual
!=	Distinto
>=	Mayor o igual
<=	Menor o igual
>	Mayor
<	Menor

operadores de asignación

Son:

Operador	Ejemplo
+=	Suma y asignación
-=	Resta y asignación
*=	Producto y asignación
/=	División y asignación
%=	Resto y asignación

Mensajes

¿qué son?

Se trata de ventanas que desde el código se lanzan al usuario para hacer que éste reaccione ante una situación o nos informe ante una duda. Realmente todos los mensajes se obtienen a través del objeto **window** (véase más adelante).

alert

Es el mensaje más usado. Saca un mensaje por la pantalla el cual sólo deja la posibilidad de aceptarle. Su uso es mostrar información al usuario pero resaltándola de la página. Su sintaxis es:

```
alert(texto_del_mensaje);
```

prompt

En este caso se trata de una ventana que pide entrar datos al usuario. De modo que esta función devuelve un valor que se puede usar en el código si es asignado a una variable. Su sintaxis es:

```
prompt(texto_del_mensaje,valor_por_defecto);
```

El segundo parámetro (valor por defecto) no es obligatorio incluirle y permite asignar un valor al cuadro de texto en el que el usuario tendrá que introducir información.

Ejemplo de uso de **prompt**:

```
respuesta=prompt("¿Qué quieres hacer?","comer");
```

En el ejemplo, el resultado de lo que el usuario responde se almacena en la variable *resultado* y al principio la ventana contendrá el valor ***comer*** en el cuadro de texto destinado al usuario. Naturalmente, el usuario podrá variar este valor si lo desea.

El cuadro de diálogo que saca **prompt** posee dos botones, uno es el de Aceptar y el otro es el de Cancelar. Si el usuario pulsa Cancelar, la función prompt devuelve el valor nulo (**null)**.

confirm

Saca un mensaje de confirmación el cual suele tener dos botones: Aceptar y Cancelar. Sintaxis:

```
confirm(texto_del_mensaje)
```

La ventana mostrará el texto elegido (normalmente es una pregunta) y el usuario elegirá si desea aceptar o no el contenido. **Confirm** devuelve un valor **true** en el caso de que el usuario acepte el mensaje, y **false** si no lo hace.

estructuras condicionales

introducción

Las estructuras de este tema son sentencias que permiten tomar decisiones dentro del código a fin de devolver un resultado u otro dependiendo de una determinada circunstancia que es la que se evalúa.

instrucción *if*

La instrucción **IF** realiza lo que se denomina un sí lógico. Su forma es:

```
if(condición) {
  ..código que se ejecuta si la condición es cierta
}
```

También admite esta otra:

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

Se admite dentro de una instrucción **IF**, colocar otra instrucción **IF**. A esto se le llama "anidar" condiciones **if**.

bucle while

Un bucle es una estructura de programación que permite repetir sentencias hasta que se cumpla una determinada condición. Su forma es:

```
while(condicion){
... sentencias que se ejecutan mientras la condición se cumpla
}
```

Ejemplo:

```
var x=1;
while(x<11)
{
   document.write(x," ");
   x++;
}
// Sale 1 2 3 4 5 6 7 8 9 10
```

bucle for

Su efecto es muy similar a la anterior estructura. Permite ejecutar una serie de sentencias hasta que se cumpla una determinada condición.. Su estructura es:

```
for(valor_inicial; condición; actualización)
{
  ..sentencias que se ejecutan mientras la condición se cumpla
}
```

Ejemplo:

```
for(x=1;x<11;x++)
{
   document.write(x," ");
}
// Sale 1 2 3 4 5 6 7 8 9 10
```

break

Es una instrucción que hace que el navegador que se la encuentra, abandone inmediatamente el bucle en el que está inmerso.

continue

Es parecida a la anterior, sólo que en lugar de abandonar el bucle, lo que hace el navegador es dejar de leer las siguientes instrucciones del bucle y saltar al principio del mismo.

instrucción switch

Esta instrucción permite un mayor control sobre las condiciones, lo malo es que se incluyó en la versión 1.2 de JavaScript por lo que sólo los navegadores con versión 4 o posterior los pueden usar.

Su sintaxis es:

```
switch (objetodeanálisis) {
   case valor1: ..instrucciones
   case valor2:...instrucciones
   ....
   default: instrucciones
}
```

Switch funciona de esta forma: en los paréntesis se coloca una expresión a evaluar, y en cada apartado case, se coloca un posible valor de la expresión. Los valores que se cumplan harán que se ejecuten las instrucciones que les siguen. En caso de que no cumpla ninguna se ejecutarían las correspondientes al **default** (no es obligatorio poner este apartado). Ejemplo:

```
var x=12;

switch(x) {
  case 4:document.write("Es cuatro");break;
  case 8:document.write("Es ocho");break;
  case 12:document.write("Es doce");break;
  default: document.write("No es ninguna de las anteriores")
}
```

Hace falta poner la instrucción **break** ya que de otro cuando se encuentra el valor que cumple la expresión, se ejecuta su **"case"** y los **"case"** siguientes.

Funciones y Objetos

funciones

introducción

Como en otros muchos lenguajes, en Javascript se pueden utilizar funciones. Las funciones son una serie de instrucciones que realizan una determinada tarea. A las funciones se las pone un nombre que luego puede ser utilizado en el código de la página.

definición de una función

Antes de poder usar una función en el código de la página, se la debe definir; es decir, se debe indicar qué operaciones son las que debe hacer la función. La definición de la función es:

```
function nombredelafunción(argumento1, argumento2,...)
{
 instrucciones que debe realizar la función
}
```

El código que está encerrado entre llaves indica lo que realiza la función (por ejemplo mostrar un mensaje de ayuda), cada vez que desde el código se llame a la función, ésta realizará sus instrucciones.

Por otro lado los argumentos son variables que algunas funciones necesitan para realizar su tarea

llamar a una función

Para usar (invocar) una función en el script, basta con poner su nombre seguido de los paréntesis.

Ejemplo:

```
function error() {
    document.write("<b>Ocurrió un error</B><BR>");
}
```

En este caso se define una función que muestra texto en la posición actual del cursor. Para utilizar esta función desde el código sería:

```
error();
```

Lo cual mostraría el mensaje de error. Otro ejemplo (página HTML completa):

```
<HTML>
<HEAD>
  <TITLE>Titulo</TITLE>
```

```
<SCRIPT LANGUAGE="JavaScript">
<!--
  function doblar(valor) {
    return valor * 2;
//-->
</SCRIPT>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
 document.write("El doble de 100 es ",doble(100));
</SCRIPT>
</BODY>
</HTML>
```

En este caso la función **doblar()**, posee un argumento (**valor**), el cual es necesario ya que esta función calcula el doble de ese número. Además esta función devuelve un valor que el código puede utilizar, eso lo realiza la instrucción **return**, la cual sirve para regresar un valor que el código puede mostrar o asignar a una variable.

funciones predefinidas

JavaScript trae consigo muchas funciones predefinidas. Señalamos aquí algunas de las más importantes:

- **eval(*textoCódigo*).** La función **eval** tiene un único parámetro que es una cadena de texto. Esta función hace que el texto sea interpretado como si fuera código normal de JavaScript. Ejemplo:

```
eval("alert('Hola')");
```

- **parseInt(*textoNúmero, base*).** Convierte el texto (que debe tener cifras numéricas) a formato de número. El segundo parámetro es opcional y representa la base del número, ejemplo:

```
alert(parseInt("110011",2));//Sale 51
```

  Si la conversión no es posible, devuelve el valor **NaN** *(Not a Number)* que indica que la variable numérica posee un valor inválido

- **parseFloat(*textoNúmero*).** Convierte el texto (que debe tener cifras numéricas) a formato de número con decimales.

- **escape(*texto*).** Muestra el código ASCII de los símbolos del texto. Cada número en el resultado va precedido del símbolo **%** y el código ASCII sale en forma Hexadecimal.

- **unescape(*texto).*** Hace justo lo inverso del anterior. Devuelve los códigos que representan los códigos ASCII en forma de texto que se le pasa como parámetro. Ejemplo:

```
document.write(unescape("%21%B1"));
// Escribe: ¡·
```

- **isNaN(*expresión*).** Devuelve **true** si la expresión tiene un contenido no numérico.

objetos

Los objetos son una de las bases fundamentales de JavaScript. Un objeto es una agrupación de variables, que en ese caso se llaman **propiedades**, y de funciones, las cuales se llaman **métodos**. Las propiedades definen las características de los objetos y los métodos las operaciones que podemos hacer con él.

JavaScript posee muchos objetos predefinidos y también permite crear nuestros propios objetos.

propiedades

Como ya se comentó anteriormente, los objetos poseen propiedades asociadas, para acceder a ellas se utiliza el punto, en esta forma:

```
objeto.propiedad
```

Ejemplo, un objeto llamado **miordenador** que representa a un ordenador:

```
miordenador.marca="HP";
miordenador.procesador="pentium III 900 Mhz"
miordenador.ram=64;
```

Las propiedades pueden ser de distintos tipos de datos; en el ejemplo la propiedad **ram** es numérica, mientras que **marca** es de tipo texto.

métodos

Los métodos son funciones asociadas a los objetos. Su uso es:

```
objeto.metodo()
```

Donde el método además puede poseer parámetros.

operación new

La instrucción **new** sirve para crear nuevos objetos en el tiempo de ejecución del código JavaScript. Ejemplo:

```
miordenador = new ordenador("HP", "Pentium III", 64);
```

En este caso **mi ordenador** es un nuevo objeto de tipo **ordenador**, al indicar el tipo de objeto que es, se le pueden pasar parámetros para rellenar sus propiedades.

objetos predefinidos

string

El objeto **string** sirve para manejar cadenas de texto. Cada vez que creamos una variable de cadena, en realidad estamos creando una variable de tipo **string**. Por lo tanto no será necesaria su declaración.

Métodos de String

- **anchor(*nombre*).** Crea un marcador en el texto.
- **big().** Muestra la cadena de caracteres con una fuente grande.
- **blink().** Muestra el texto de modo intermitente.
- **bold().**Muestra el texto en negrita.
- **charAt(*n*).** Muestra el carácter situado en la posición ***n*** de la cadena.
- **fixed().** Muestra la cadena en fuente no proporcional.
- **fontcolor(*color*).** Determina el color del texto.
- **fontsize(*n*).** Muestra el texto en el tamaño ***n***, donde ***n*** es un número del 1 al 7 (los 7 tamaños estándar).
- **indexOf(*cadenaInterna,inicio)***. Devuelve la posición de la cadena interna en el texto, teniendo en cuenta que el primer carácter es el número 0. El primer parámetro es el texto que se busca; el segundo es opcional e indica desde qué posición del texto comenzamos a buscar. Si no se encuentra la cadena interna, se devuelve el valor –1. Ejemplo:

```
var cadena="Miguel Indurain"
var subcadena="Indurain"
alert(cadena.indexOf(subcadena))
/*El resultado será 7, ya que la primera posición del
  texto (en este caso la 'M') es la 0.
```

- **italics().** Muestra el texto en cursiva.
- **lastIndexOf(*cadenaInterna,inicio*).** Idéntico a **indexOf** sólo que en este caso cuenta la última vez que aparece la cadena (en lugar de la primera vez como hace **indexOf**).
- **link(*URL*).** Crea un hipervínculo en la cadena de texto, el parámetro ***URL*** indica el destino del vínculo.
- **small().** El texto se muestra con fuente pequeña.
- **strike().** Subraya el texto.

- **sub().** El texto va en subíndice.
- **substring(*x,y*).** Muestra el fragmento de texto que va desde la posición **x** a la posición **y**. Ejemplo:

```
 var cadena="Miguelón Indurain";
 var subcadena="Indurain";
 alert(cadena.substring(5,11));
//Sale lón In
```

- **sup().** Superíndice.
- **toLowerCase().** Convierte la cadena a minúsculas.
- **toUpperCase().**Convierte la cadena a mayúsculas.

Propiedades de string

- **length.** Almacena el tamaño de la cadena de texto.

Math

El objeto **Math** tiene propiedades y métodos que representan valores matemáticos.

métodos de math

- **abs(*n*).** Calcula el valor absoluto de n.
- **acos(*n*).** Calcula el arco coseno de n.
- **asin(*n*).** Calcula el arco seno de n.
- **atan(*n*).** Calcula el arco tangente de n.
- **ceil(*n*).** Redondea n a su valor superior.
- **cos(*n*).** Calcula el coseno de n.
- **exp(*n*).** Calcula e^n.
- **floor(*n*).** Redondea n a su valor inferior.
- **log(*n*).** Calcula el logaritmo de n.
- **max(x,y).** Devuelve el mayor valor de x o y.
- **min(x,y).** Devuelve el menor valor de x o y.
- **pow(x,y).** Devuelve el x^y.
- **random().** Genera un número aleatorio entre 0 y 1. Ejemplo:

```
alert(Math.random())
// Podría devolver por ejemplo 0.239230812349
```

- **round(n).** Redondea n a el número más próximo.
- **sin(n).** Calcula el seno de n.
- **sqrt(n).** Calcula la raíz cuadrada de n.
- **tan(n).** Calcula la tangente de n.

propiedades de math

- **E.** Devuelve la constante de Euler o número **e.**
- **LN2.** Devuelve el logaritmo neperiano de 2.
- **LN10.** Devuelve el logaritmo neperiano de 10.
- **LOG2E.** Logaritmo en base 2 de **e**.
- **LOG10E.** Logaritmo en base 10 de **e.**
- **PI.** Devuelve el número PI.
- **SQRT1_2.** Raíz cuadrada de 0,5.
- **SQRT2.** Raíz cuadrada de 2.

objeto Date()

Este objeto representa fechas y horas. JavaScript no permite trabajar con fechas anteriores a 1970, ya que desde ese momento es cuando comienza a contar las fechas en milisegundos. En los meses, el mes 0 será Enero, y el mes 11 es Diciembre. Los días de la semana y del mes se cuentan desde el número 1 (Jueves = 4).

Para crear una variable de fecha se puede hacer de esta manera:

```
fecha=new Date();
//crea un nuevo objeto de fecha cuyo valor inicial serán
//la fecha y hora actuales
```

Si se desea iniciar una variable de fecha con valores distintos a la fecha actual, se debe:

```
fecha=new Date(año, mes, día, hora, minutos, segundos");
```

Todos los parámetros se pasan en forma de números. También se puede crear un objeto de tipo fecha asignando el número de milisegundos desde 1970.

métodos de los objetos date()

- **getDate().** Devuelve el día del mes.
- **getDay().** Devuelve el día de la semana en forma de número.

- **getFullYear().** Devuelve el año, pero en forma de 4 números. Con esto se asegura la compatibilidad con el año 2000. Está función se añadió al JavaScript 1.3, por lo que ciertos navegadores no podrán usarla.
- **getHours().** Devuelve la hora.
- **getMinutes().** Devuelve los minutos.
- **getMonth().** Devuelve el mes (con números del 0 al 11).
- **getSeconds().** Devuelve los segundos.
- **getTime().** Devuelve el número de milisegundos de la fecha, empezando a contar desde 1970.
- **getTimezoneOffset().** Devuelve la diferencia en minutos entre la zona horaria actual y la hora solar (GMT).
- **getYear().** Devuelve el año.
- **setDate(*valor*).** Establece el día del mes.
- **setFullYear(*valor*).** Establece el año (con cuatro cifras).
- **setHours(*valor*).** Establece la hora.
- **setMinutes(*valor*).** Establece los minutos.
- **setMonth(*valor*).** Establece el mes (con un número del 1 al 11).
- **setSeconds(*valor*).** Establece los segundos.
- **setTime(*valor*).** Establece la fecha con el número de milisegundos desde el 1 de Enero de 1970.
- **setYear(*valor*).** Establece el año.
- **toLocaleString().** Devuelve la fecha en formato de texto, según las especificaciones de la zona horaria del ordenador.

objeto array

Los arrays (matrices) son elementos indispensables en la programación de ordenadores. Un array es un conjunto de datos agrupados. Para acceder a cada elemento individual de el array se usa un número de índice, el primer elemento tendrá el índice 0. En el caso de los arrays de JavaScript, su uso es muy eficaz y mucho más libre que en los lenguajes formales (como Pascal por ejemplo).

Para crear un array se hace:

```
nombrearray = new Array()
```

Esto crea un array de tamaño indeterminado.

Para rellenar los valores del array:

```
nombrearray[0] = valor;
nombrearray[1] = valor;
...
nombrearray[n] = valor;
```

Tras asignar valores el array se hace más grande. También se puede especificar su tamaño al crearle:

```
nombreArray= new Array(tamaño)
```

O incluso asignar valores en la propia creación del array. Ejemplo:

```
equipos= new Array("Real Madrid", "F. C. Barcelona");
```

Además un array puede tener distintos tipos de datos:

```
miOrdenador = new Array("HP", "Pentium III", 64);
```

Y cada elemento de un array puede ser otro array:

```
elemento = new Array(8);
elemento[3] = new Array(5);
```

métodos del objeto array

- **concat(*array*).** Agrupa dos arrays. Disponible desde la versión 1.2. Ejemplo:

```
a = new Array(12, 3, 5);
b = new Array("Hola", "Adios");
c = a.concat(b);
//c es el array (12, 3, 5, "Hola", "Adios")
```

- **join().** Saca una cadena de texto que contiene todos los elementos del array:

```
a = new Array("Rojo", "Azul", "Verde");
b = a.join();
//b contiene "Rojo,Azul,Verde"
```

- **reverse().** Invierte el orden del los elementos de un array. El primero pasa a ser el último y viceversa.

- **Sort().** Obtiene la matriz de manera ordenada.

Propiedades del objeto Array

- **length.** Cuenta el número de elementos del array.

objetos del navegador

introducción

Los objetos del navegador son todos aquellos objetos que el navegador pone a nuestra disposición para poder modificar los elementos de las páginas.

El problema es que Explorer y Navigator utilizan modelos de objetos distintos. En este documento sólo se comentan los objetos y propiedades comunes. Los objetos son:

navigator

Objeto que representa al navegador con el que el usuario está mostrando la página. Gracias a este objeto se puede averiguar la marca y versión del navegador y utilizar esta información en el código.

propiedades

- **appCodeName**. Nombre del código del navegador. Todos los navegadores devuelven la cadena **Mozilla**. Por lo que esta propiedad no es interesante.
- **appName**. Nombre del navegador. Podrá ser: **Microsoft Internet Explorer** o **Netscape.**
- **appVersion**. Versión del navegador. La cadena que devuelve esta propiedad indica la versión completa. Ejemplos:

Cadena devuelta por appVersion	Navegador
4.0 (compatible; MSIE 6.0; Windows NT 5.0)	Explorer 6 para Windows 2000
4.0 (compatible; MSIE 5.5; Windows NT 5.0)	Explorer 5.5 para Windows 2000
4.0 (compatible; MSIE 6.0; Windows 2000)	Opera 6 para Windows 2000
5.0 (Windows; es-ES)	Netscape 6 en español
4.75 [en] (Windows NT 5.0, U)	Netscape 4.75
4.5 [es] C-CCK-MCD (WinNT;I)	Netscape 4.5

- **language** o **browserLanguage. language** es una propiedad que sólo funciona con Netscape, mientras que **browserLanguage** sólo funciona con Explorer. Pero ambas devuelven el lenguaje del navegador. En ambos casos las dos primeras letras que devuelven son el lenguaje del navegador (es=Español; en=Inglés, por ejemplo).
- **platform**. Contiene el tipo de sistema operativo del ordenador del usuario. **Win32** indicaría un sistema Windows 95/98/Me/NT/2000. Otros valores son: **Win16** (Windows 3.1 y anteriores) **Mac68x** (McIntosh clásico) **MacPPC**

(McIntosh Power PC) y varios resultados que empiezan por **Unix** (sistemas Unix)**.**

- **userAgent**. Devuelve información sobre la versión, código y nombre del navegador. Su salida no aporta nada nuevo respecto a las anteriores propiedades. Ejemplos:

Cadena devuelta por userAgent	Navegador
Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)	Explorer 6 para Windows 2000
Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)	Explorer 5.5 para Windows 2000
Mozilla/4.0 (compatible; MSIE 5.0; Windows 2000) Opera 6.0 [es]	Opera 6 para Windows 2000
Mozilla/5.0 (Windows; U; Windows NT 5.0; es-ES; rv:0.9.4) Gecko 20011128 Netscape6/6.21	Netscape 6.21 en español
Mozilla/4.75 [en] (Windows NT 5.0, U)	Netscape 4.75
Mozilla/4.5 [es] C-CCK-MCD (WinNT;I)	Netscape 4.5

screen

Objeto disponible desde la versión 1.2 de JavaScript (navegadores 4 o superiores). Permite acceder a las propiedades de la pantalla del usuario.

propiedades

- **width** y **height.** Respectivamente, anchura y altura total de la pantalla.
- **availWidth** y **availHeight.** Respectivamente, anchura y altura disponible en la pantalla tras restar la barra de tareas del sistema operativo. Esta medida no es muy exacta ya que no tiene en cuenta la personalización del usuario.
- **colorDepth.** Número de bits por píxel que utiliza la pantalla.

window

Este objeto, representa a la ventana en la cual se ve la página web. En el caso de que la página tenga marcos, se generan tantos objetos window como marcos haya.

propiedades

- **closed.** Valor booleano que indica si una ventana ha sido cerrada.
- **defaultStatus.** Texto que la barra de estado mostrará cuando se cargue la página web (texto por defecto de la barra de estado).
- **frames.** Array que representa a todos los marcos de la ventana.

- **history.** Es un objeto (se verá más adelante) que representa las enlaces a las páginas a las que el visitante había accedido antes de llegar a la ventana actual.
- **location**. Objeto que almacena información sobre el URL de la página actual.
- **name.** El nombre de la ventana.
- **parent.** Ventana "padre" de la actual. Si la actual es un marco, parent será la página con etiqueta <FRAMESET>.
- **self**. Se refiere a la propia ventana activa.
- **top.** Ventana superior del navegador.
- **status**. Mensaje de la barra de estado.
- **window.** Igual que **self**.

métodos

- **open(*URL,nombreVentana,opcionesVentana*)**. Abre una nueva ventana cuyo contenido se especifica por la URL a una página (este parámetro puede quedar vacío ""), se indica un nombre y, opcionalmente, una serie de opciones entre comillas y separadas por comas. Estas opciones son (Netscape tiene algunas opciones más que aquí no hemos listado):
  - **toolbar.** Indica con **yes** o **no** si se muestra la barra de herramientas.
  - **location.** Muestra o no la barra de dirección.
  - **directories.** Muestra o no los botones de directorio.
  - **status.** Permite mostrar u ocultar la barra de estado.
  - **menubar**. Mostrar o no la barra de menús.
  - **scrollbars.** Indica si se muestran las barras de desplazamiento.
  - **resizable.** Permite ajustar el tamaño de la ventana.
  - **width.** Ancho de la ventana en píxeles.
  - **height.** Altura de la ventana en píxeles.
  - **copyHistory**. Permite copiar el historial de páginas recorridas a la nueva ventana.

  Ejemplo:

```
nuevaVentana=open("","nueva","toolbar=no,menubar=no,
scrollbars=no,location="no,width=180,height=60");
```

- **close()**. Cierra la ventana.

- **blur()**. Hace la ventana deje de estar activa (a esta acción se la llama perder el foco).
- **focus()**. Hace que la ventana sea la ventana activa (la que tiene el "foco").
- **setInterval(*expresiónjavascript,milisegundos*)**.. Crea un temporizador. El temporizador es un reloj que cada ciertos milisegundos (los que se indiquen como segundo parámetro, realiza la tarea indicada en el argumento expresión. La expresión es código JavaScript el cual se coloca entre comillas, normalmente este código es simplemente la invocación a una función. este función devuelve un número de temporizador (ya que se pueden crear varios) el cual deberá almacenarse en una variable para posteriores manipulaciones. Ejemplo:

```
var tempID;
tempID=setInterval("dibujaCirculo();",1000ms);
//Hace que cada segunda se llame a la función
dibujaCirculo().
```

- **clearInterval(*idTemporizador*)**. Cancela el tiempo de espera establecido mediante ***setInterval*.**. Al llamar a este método hay que indicar como parámetro el número de temporizador que se desea detener.
- **setTimeout(*expresiónjavascript,milisegundos*)**. Muy similar a ***setInterval***. Cuando pasan los milisegundos invocados, se ejecuta el código del parámetro ***expresiónJavascript*** (el cual va entre comillas). La diferencia con ***setInterval*** es que en este caso sólo se usa el código una vez, no continuamente como ocurre con ***setInterval.*** El método devuelve un número que debe almacenarse en una variable.
- **clearTimeout(*idTimeOut*).** Anula el temporizador establecido anteriormente con ***setTimeOut.***
- **alert(), confirm(), prompt().** Métodos de mensajes (vistos en tema anterior).

location

Objeto incluido dentro del objeto window. Almacena información sobre la localización de la página de la ventana y por tanto permite cambiar dinámicamente la página web que se está mostrando.

propiedades

- **href**. Especifica la dirección URL de la ventana (por ejemplo: *http://www.uva.es/problemas/ex1.htm#marca1"*)
- **hostname**. Especifica la parte del URL en la que va el nombre del host *(www.uva.es*)
- **host**. Idéntico al anterior, sólo que al final se indica el número de puerto utilizado *(www.uva.es:80*)

- **pathname**. Especifica la parte del URL en la que va la ruta al recurso (en el ejemplo: */problemas/ex1,htm"*)
- **port**. Puerto utilizado para mostrar la página (generalmente el 80).
- **hash**. Indica qué marcador de la página se utilizó al abrir la misma, si no se usó ninguno aparece vacío (en el ejemplo sería *marca1*).
- **protocol**. Parte referida al protocolo de la URL (en el ejemplo *http*).
- **search**. La parte de una URL que va detrás del signo **?**. Sólo ciertas páginas llevan este signo (en concreto las páginas que llaman a CGIs).

métodos

- **replace(*URL*)**. Carga una nueva página en la ventana actual indicando su URL entre comillas y además también reemplaza a la página anterior en la lista del historial. Esto último es la única diferencia entre usar este método y cambiar la dirección directamente en la propiedad **href** del objeto **location.**
- **reload().** Hace que se vuelva a recargar la página.

document

Este objeto representa al contenido de una página web. Está incluido dentro del objeto **window**.

propiedades

- **bgColor**. Color del fondo
- **fgColor**. Color del texto.
- **linkColor**. Color de los enlaces normales.
- **vlinkColor**. Color de los enlaces visitados
- **alinkColor**. Color de los enlaces activos.
- **images**. Array que contiene todas las imágenes del documento. El índice del array empieza por 0, la imagen **document.images[0]** será la primera que se utilizó en el código. Ejemplo:

```
document.images[0].src =”grafico2.gif”
//Hace que la primera imagen muestre el gráfico
//“grafico2.gif”
```

También se puede usar cada imagen en lugar de por el número, por el nombre (atributo NAME de la etiqueta IMG). Ejemplo:

```
document.images[“imagen1”}.src=”grafico2.gif”
```

```
//imagen1 debe ser el nombre exacto de la imagen a
//cambiar.
```

- **links.** Array que contiene todos los enlaces que usan el atributo HREF. El orden de los enlaces en la matriz es el orden de uso en el código (el primer enlace en el código será **document.links[0].**) Ejemplo:

```
document.links[0].href =”http://www.hola.com”
//hace que el primer enlace del documento apunte a la
dirección www.hola.com
```

- **lasModified**. Fecha de última actualización del documento en forma de cadena de texto. Algunos servidores no proporcionan este dato.
- **URL**. URL del documento. Es idéntico a utilizar **location.href**.
- **cookie**. Escribe o lee el archivo de cookies de la página web. Un archivo de cookies sirve para guardar información sobre el usuario en su propia máquina, con esta propiedad se permite hacerlo

métodos

- **clear()**. Borra el documento.
- **write(*textoHTML*)**. Escribe el texto indicado en el documento. Ese texto puede contener si se desea etiquetas HTML
- **writeln(*textoHTML*)**. Lo mismo que la anterior, sólo que esta añade un salto de línez tras escribir el texto.
- **close()**. Cierra el documento.

history

Objeto que representa a las direcciones de las páginas que se almacenan en el historial del navegador. Este objeto está dentro del objeto **window.**

propiedades

- **length**. Número de páginas almacenadas actualmente en el historial.

métodos

- **back()**. Hace que la ventana muestre la página visitada anteriormente.
- **forward()**. Hace que la ventana muestre la página siguiente.
- **go(*nº*).** Hace que se muestre la página del historial indicada con un número. De modo que **history.go(-1)** muestra la página anterior y **history.go(-3)** hace que se muestre la página antepenúltima.

image

Objeto que representa una imagen en el documento definida con la etiqueta HTML <IMG> o con el código JavaScript **new Image**. Todas las imágenes del documento están contenidas en la matriz de imágenes **document.images** (ver anteriormente).

propiedades

Son las mismas que los atributos de la etiqueta IMG del HTML, por eso sólo se comentan:

- **border**
- **height**
- **hspace**
- **lowsrc**
- **name**
- **src**
- **vspace**
- **width**

Ejemplo:

```
var imagenAtras = new image(120,87);
//nueva imagen con ancho=120 y altura=87
imagenAtras.src="dibujogris.gif";
//la imagen muestra el archivo dibujogris.gif
```

link

Representa cada enlace de la página creado con la etiqueta A. El array **document.links** contiene todos los enlaces del documento, de forma que **document.links[0]** será el primer enlace escrito en el código.

propiedades

- **href**. Especifica la dirección URL de Internet del enlace (por ejemplo: *http://www.uva.es/problemas/ex1.htm#marca1"*)
- **hostname**. Especifica la parte del URL en la que va el nombre del host *(www.uva.es*)
- **host**. Idéntico al anterior, sólo que al final se indica el número de puerto utilizado *(www.uva.es:80*)
- **pathname**. Especifica la parte del URL en la que va la ruta al recurso (en el ejemplo: */problemas/ex1,htm"*)

- **port**. Puerto utilizado para mostrar la página (generalmente el 80).
- **hash**. Indica qué marcador de la página se utilizó al abrir la misma, si no se usó ninguno aparece vacío (en el ejemplo sería *marca1*).
- **protocol**. Parte referida al protocolo de la URL (en el ejemplo *http*).
- **search**. La parte de una URL que va detrás del signo **?**. Sólo ciertas páginas llevan este signo (en concreto las páginas que llaman a CGIs).
- **target**. Indica el destino del enlace (propiedad TARGET de la etiqueta A), interesante sobre todo si la ventana tiene marcos.

eventos

introducción

Un evento es un suceso que ocurre cuando el usuario realiza alguna acción. Por ejemplo cuando el usuario pasa el ratón encima de un objeto de la página, cuando pulsa una tecla,... Incluso algunos eventos no los produce el usuario, sino el navegador, como por ejemplo la carga de la página.

Los eventos se colocan en etiquetas de HTML (no todas tienen capacidad para capturar eventos y además nuevamente no son las mismas en Netscape que en Explorer), de esta forma:

```
<etiquetaHTML atributos...
nombredeEvento="expresiónJavaScript">+
```

De tal forma que cuando se produce el evento en cuestión, se ejecuta el código en JavaScript que está entre comillas. Ejemplo:

```
<A HREF="cities.htm" onMouseClick="alert('Pulsaste!');">
//Cuando el usuario pulsa en el mensaje, aparece un cuadro
//de alerta que pone "Pulsaste!"
```

lista de eventos

Los eventos son iguales en Netscape y en Explorer, sin embargo no se aplican a los mismos objetos, hay más objetos en Explorer que en Netscape que permiten capturar eventos. La lista sólo incluye los compatibles en ambos.

onClick

Se produce cuando el usuario hace clic en el objeto. Sólo los vínculos y los botones de los formularios permiten capturar este evento en ambos navegadores.

onDblClick

Se genera cuando el usuario hace doble clic con el ratón. Sólo los vínculos admiten este evento en ambos navegadores.

onMouseOver

Se produce cuando el usuario pasa el cursor por encima del objeto. Sólo los vínculos permiten este evento en ambos navegadores.

onMouseOut

Ocurre cuando el usuario abandona el objeto. Sólo los vínculos permiten este evento en ambos navegadores.

onMouseDown

Se produce cuando el usuario mantiene pulsado el botón principal del ratón a la vez que se sitúa encima de un objeto. Funciona en los vínculos, en los botones y en las imágenes.

onMouseUp

Ocurre cuando el usuario levanta el botón del ratón. Funciona en los mismos casos que el anterior.

onMouseMove

Sucede cuando el usuario mueve el ratón. En Netscape ninguna etiqueta lo admite.

onKeyDown, onKeyPress y onKeyUp

Ocurren respectivamente cuando el usuario pulsa una tecla, cuando la mantiene pulsada y cuando la suelta. Tienen poca utilidad y pocas etiquetas los admiten.

onLoad

Se produce cuando la página se está cargando. La etiqueta BODY es la idónea para este evento.

onUnload

Se produce cuando la página se está descargando, porque se está cargando otra o porque se cierra el navegador. La etiqueta BODY es la idónea para este evento.

onResize

Ocurre cuando se cambia el tamaño de la ventana. La etiqueta BODY o la FRAME son las que manejan este evento.

onBlur

Se produce cuando un objeto pierde el foco (deja de ser el objeto activo). La etiqueta BODY, los botones, cuadros de formulario y los enlaces admiten esta propiedad.

onFocus

Sucede cuando un objeto gana el foco (pasa a ser el objeto activo). La etiqueta BODY, los botones, cuadros de formulario y los enlaces admiten esta propiedad.

onAbort

Se produce si el usuario pulsa el botón ***Detener*** mientras se estaba cargando una imagen. La etiqueta IMG es la que maneja este evento.

onError

Se produce cuando ocurre un error. Casi todas lo permiten, no obstante su uso no parece muy interesante.

onChange

Se produce cuando el usuario cambia el contenido de un cuadro de texto de un formulario.

onSelect

Ocurre cuando el usuario selecciona texto de un cuadro del formulario.

onSubmit

Ocurre cuando un formulario es enviado a su servidor. Es pues un evento de la etiqueta FORM.

onReset

Sucede cuando un formulario es anulado mediante su botón **Reset**. Es pues un evento de la etiqueta FORM.