MANUAL DE JAVASCRIPT

de Jose Antonio Rodríguez

Publicado en la página web www.internetmania.net

Adaptación del curso "Javascript desde cero", publicado en:
www.ciudadfutura.com/javascriptdesdecero

INDICE:

1) Introducción al lenguaje JavaScript:
   A) Programas, lenguajes, scripts...
   B) El Javascript....
   C) Variables, datos, objetos...
   D) La ejecución de los scripts.

2) Elementos del lenguaje:
   A) Los datos
   B) Las variables
   C) Los Objetos
   D) Los Arrays
   E) Las funciones

3) Los operadores:
   A) Operadores aritméticos
   B) Operadores binarios
   C) Operadores lógicos
   D) Operadores varios
   E) Funciones Globales en Javascript
   F) Expresiones regulares

4) La gramática:
   A) Gramática JavaScript: Condicionales
   B) Gramática JavaScript: Selección múltiple
   C) Gramática JavaScript: Bucles
   D) Gramática JavaScript: Ruptura de bucles

5) Los objetos:
   A) Objetos en JavaScript: Array
   B) Objetos booleanos: Verdadero o Falso
   C) Objeto Function
   D) Objeto Number
   E) Objeto Object
   F) Objeto Regular Expresion
      a) Métodos RegExp: Compile
      b) Métodos RegExp: Exec
      c) Métodos RegExp: Test
      d) RegExp
   G) Objeto String
      a) Métodos de String: anchor
      b) Métodos de String: big
      c) Métodos de String: blink
      d) Métodos de String: bold
      e) Métodos de String: charAt
      f) Métodos de String: charCodeAt
      g) Métodos de String: concat
      h) Métodos de String: fixed
      i) Métodos de String: fontcolor

j) Métodos de String: fontsize
k) Métodos de String: fromCharCode
l) Métodos de String: indexOf
m) Métodos de String: italics
n) Métodos de String: lastIndexOf
o) Métodos de String: link
p) Métodos de String: match
q) Métodos de String: replace
r) Métodos de String: search
s) Métodos de String: slice
t) Métodos de String: small
u) Métodos de String: split
v) Métodos de String: strike
w) Métodos de String: sub
x) Métodos de String: substr
y) Métodos de String: substring
z) Métodos de String: sup
aa) Métodos de String: toLowerCase
bb) Métodos de String: toUpperCase

H) Objeto Date
a) Métodos de Date: getDate()
b) Métodos de Date: getDay()
c) Métodos de Date: getFullYear()
d) Métodos de Date: getHours()
e) Métodos de Date: getMilliseconds()
f) Métodos de Date: getMinutes()
g) Métodos de Date: getMonth()
h) Métodos de Date: getSeconds()
i) Métodos de Date: getTime()
j) Métodos de Date: getTimezoneOffset()
k) Métodos de Date: getYear()
l) Métodos de Date: Object.toString()
m) Métodos de Date: Object.valueOf()
n) Métodos de Date: parse()
o) Métodos de Date: setDate()
p) Métodos de Date: setFullYear()
q) Métodos de Date: setHours()
r) Métodos de Date: setMilliseconds()
s) Métodos de Date: setMinutes()
t) Métodos de Date: setMonth()
u) Métodos de Date: setSeconds()
v) Métodos de Date: setTime()
w) Métodos de Date: setYear()
x) Métodos de Date: toGMT()
y) Métodos de Date: toLocaleString()
z) Métodos de Date: toUTCString()

I) Objeto Math
a) Métodos Math: abs
b) Métodos Math: acos
c) Métodos Math: asin

d) Métodos Math: atan
e) Métodos Math: atan2
f) Métodos Math: ceil
g) Métodos Math: cos
h) Métodos Math: exp
i) Métodos Math: floor
j) Métodos Math: log
k) Métodos Math: max
l) Métodos Math: min
m) Métodos Math: pow
n) Métodos Math: random
o) Métodos Math: round
p) Métodos Math: sin
q) Métodos Math: sqrt
r) Métodos Math: tan

6) Ejemplos JavaScript:
A) Operadores
B) Media Aritmética
C) Saludo
D) Array aleatorio
E) Comprobar E-mail 1
F) Comprobar E-mail 2
G) Buscar en un Array
H) Extraer Subcadena
I) Creando Objetos

7) Aplicaciones JavaScript HTML:
A) El Reloj en pantalla
B) Fecha de Actualización
C) Menús Desplegables (IE)
D) Formularios de Correo
E) Personalizar Colores
F) Persianas
G) Rollover
H) Información del Navegador
I) Esquema desplegable (IE)
J) Botón más/menos
K) Password 1
L) Título de página animado
M) Bloque fijo
N) Paisaje Nevado 1
O) Paisaje Nevado 2
P) Estrella Navideña 1
Q) Estrella Navideña 2
R) Buscador en la Página
S) Página de Inicio
T) Carrusel de Imágenes

1) Introducción al lenguaje JavaScript:

A) Programas, lenguajes, scripts...

Antes de comenzar: Nada de asustarse por la jerga técnica. Para quién comienza desde cero, estas cosillas son lo primero que debiera conocer: Que es eso de programa, lenguaje, scripts o guiones....

Lo primero a aclarar es lo del **programa**, algo que realmente no es exclusivo de la informática, todos hemos programado alguna vez: al poner el vídeo para que grabe cuando no estamos en casa, cuando ponemos el despertador para que nos dé el disgusto matutino...En el caso de un programa informático los que programamos es un ordenador y los programas habitualmente se hacen escribiéndolos en un cierto **lenguaje**.

Concretando, un **programa** *no es mas que una serie de instrucciones que le damos a un sistema para que haga cosas.* En otras palabras, y en el caso que nos atañe, es decirle al ordenador como hacer una determinada tarea. Puede por tanto ser algo tan simple como decirle que sume dos números y nos diga el resultado hasta algo tan complejo como decirle que controle todo un edificio: temperatura, puertas, iluminación... En nuestro caso es algo bastante sencillo, vamos a decirle al ordenador que cuando presente nuestra página web al visitante haga cosas como poner en la página la fecha del día, hacer que una imagen se mueva de un lado a otro, responder de una determinada forma a la pulsación del ratón, etc.

Para escribir un programa no nos vale ni el castellano, ni el inglés, ni ninguno de los lenguajes que habitualmente usa el hombre para comunicarse. Para entendernos con un ordenador se utilizan los lenguajes informáticos. Existen dos grandes grupos de lenguajes a la hora de escribir un programa: **Los compilados** y **Los interpretados**.

Cuando usamos lenguajes compilados seguimos los siguientes pasos:

- Nosotros escribimos el programa (código fuente),
- Un programa lo traduce al lenguaje interno del ordenador (se compilan)
- Se obtiene así un nuevo fichero que es el que se ejecuta tantas veces como se desee.

Los programas así obtenidos son los que se almacenan en ficheros con un nombre terminado en **.exe** o en **.com**, si trabajamos en los sistemas operativos DOS o Windows. Podríamos resumir: Un programa compilado se traduce una vez y se ejecuta cuantas veces se desee. A este grupo pertenecen los lenguajes tradicionales como C, Pascal, Ada y otros.

El otro grupo es el de los lenguajes interpretados, en éstos el programa se escribe (código fuente), y el ordenador va leyendo cada instrucción, la traduce y la ejecuta. O sea, es necesario traducir el programa cada vez que se ejecuta. ¿Quién traduce las instrucciones del programa? Pues muy sencillo, otro programa: el traductor o **intérprete**. A este grupo pertenece el legendario Basic, el Perl y los llamados scripts como JavaScript y Vbscript. Los programas escritos en estos dos últimos lenguajes son

interpretados por los navegadores de Internet como Internet Explorer, Netscape Navigator, Opera.

Muy bien ya sé que es un programa, pero a la hora de la verdad ¿cómo se construyen los programas? ¿Cómo se ejecutan?. Paciencia, sigue viendo las páginas de introducción, mira por encima los ejemplos de JavaScript (deja el HTML dinámico para mas adelante) y pasa a ver la gramática. Luego te das un paseo por los objetos. Pero todo gradualmente y con paciencia.

B) El Javascript....

Escribir un programa por lo tanto es simplemente escribir instrucciones para que las ejecute el ordenador, utilizando para ello un cierto lenguaje. Es como escribir en inglés: necesitas conocer el vocabulario y la gramática del idioma de Shakespeare. En nuestro caso usamos como lenguaje el **JavaScript** y necesitas conocer sus reglas y su vocabulario. Como ya sabes se trata de un lenguaje interpretado y los programas escritos con estos lenguajes son conocidos como **scripts** o guiones. Pese a su nombre no tiene nada que ver con **Java**, este último es un lenguaje algo más complejo con el que se pueden construir programas de propósito general como podría hacerse con C++ o Visual Basic, la particularidad que tienen los programas Java es que pueden funcionar en cualquier ordenador y con cualquier sistema operativo. Las aplicaciones escritas para Internet en Java son conocidas como **applets**.

La única razón de ser de JavaScript son las páginas web. Con JavaScript no pueden construirse programas independientes, sólo pueden escribirse scripts que funcionarán en el entorno de una página Web, interpretado por un explorador, de ahí la importancia de conocer para que explorador escribimos los guiones. Y aquí viene el primer obstáculo: no todos los exploradores integran en la misma forma los guiones JavaScript.
La primera versión de JavaScript se debe a Netscape, que lo introdujo con la versión 2.0 de su explorador, posteriormente han ido surgiendo nuevas versiones habiendo sido estandarizado por la European Computer Manufacturers Asociation (**ECMA**).

Versión Exp	**Netscape Navigator**	**Microsoft Internet Explorer**
2.0	JavaScript 1.0	No lo soporta
3.0	JavaScript 1.1	JavaScript 1.0
4.0	JavaScript 1.2	JavaScript 1.2 (ECMA)
5.0	-----	JavaScript 1.5 (ECMA)

Pondríamos preguntarnos por que el esfuerzo de aprender JavaScript, ya es bastante con el HTML para construir páginas Web. En parte esto es cierto, con un buen programa editor podemos obtener una página para publicar en la red, pero esa página Web consistiría en: texto, imágenes e hipervínculos, con los atributos como colores, tipos de letra y poco más sobre los que el autor puede actuar. Pero ¿y si quisiéramos poner un menú desplegable?, ¿y si queremos que el visitante pueda mover una imagen por la pantalla? .¿y si necesitamos validar el texto entrado por el usuario?.... En resumen si queremos ir mas allá de la simple presentación de un documento en pantalla y queremos controlar al explorador no hay mas remedio que utilizar un programa. ¿Por qué en JavaScript? muy simple: es soportado por todos los exploradores, es sencillo y es el que está siendo contemplado por los organismos de normalización.

C) Variables, datos, objetos...

Para comenzar a utilizar Javascript (y cualquier lenguaje de programación) es necesario conocer algunos conceptos básicos, no podemos empezar a hacer una casa si no sabemos que existen los ladrillos. Un programa es una lista de instrucciones, pero esas instrucciones se deberán ejecutar sobre algo, si damos una instrucción *escribir* debemos especificar que es lo que hay que escribir. Es evidente pues que en las instrucciones del programa también deben figurar los datos con que trabajar. Por ejemplo el nombre de una persona es "Juan", esta palabra es un dato. El precio de una manzana en ptas. es 10, este número es otro dato. Estos datos no suelen usarse tal cual sino que se almacenan en unos elementos con nombre denominados **Variables**. En los ejemplos previos usaría una variable, *nombre*, para guardar "Juan" o *precio* para guardar el 10. Si ahora digo al ordenador que escriba *nombre* el ordenador escribirá su contenido, o sea, Juan. Un pequeño ejemplo par no aburrirnos: abre una ventana y escribe en la caja de direcciones del explorador lo que sigue:

```
javascript:nombre="juan";window.alert(nombre)
```

¡Acabas de escribir un par de instrucciones en JavaScript! (no te preocupes aún de su significado), como ves has dado un valor a la variable *nombre* y luego se ha escrito el valor de esa variable. En el ejemplo anterior también podrías haber escrito: `window.alert("Mi nombre: "+nombre).`

Mediante el símbolo + has concatenado ambas cadenas. Has usado un **operador**, algo que no es nuevo si has estudiado alguna vez matemáticas. Los operadores son símbolos usados para realizar operaciones con los datos, son los símbolos +, -, /, *, respectivamente sumar, restar, dividir y multiplicar. Como sabes estos operadores se usan con datos numéricos: 4 + 5 son 9. Estas dos instrucciones que has escrito podrían encerrarse en un bloque con un nombre**,** por ejemplo *mepresento()* y tendrías una **función**. Podríamos definir una función llamada *quehoraes()* que encierre las instrucciones necesarias para que aparezca en pantalla la hora actual. Es decir mediante las funciones creamos las órdenes que luego podremos darle al navegador para que actúe según nuestro deseo.

Hasta aquí has visto los elementos básicos existentes cualquier lenguaje de programación, pero alguna vez habrás leído que Javascript es un lenguaje que utiliza objetos a diferencia de Java que es un lenguaje orientado a objetos. ¿Y que es eso de objetos? Los **objetos** son como una extensión de las variables, una estructura que nos permite agrupar diferentes valores y nombres de funciones. Una variable numérica sólo puede contener eso, un número: 10 o 20 o 2000, una cadena tipo sólo puede guardar una serie de caracteres "ACAD", "LE18P". Un objeto va mas allá puede guardar varias cosas a la vez. ¿Suena raro?. Veamos un ejemplo muy sencillo: un rectángulo se caracteriza por la longitud de sus lados y por la forma de dibujarlo. En un programa el rectángulo se asimilaría a un objeto que tendría dos propiedades: base y altura, y un método: como_dibujarlo. Este como_dibujarlo sería una función que dibuja rectángulos. Si un programa define la variable mirectángulo como un objeto de este tipo y contiene la instrucción mirectangulo.como_dibujarlo dibujaría en pantalla un rectángulo. La ventana del explorador que tienes delante es un objeto con propiedades como: altura, anchura, nombre... y métodos como abrir, cerrar, mover...Vamos a hacer algo con este objeto que se llama *window*, para ello abre otra ventana de tu navegador y escribe en la barra de direcciones *javascript:window.close()*. ¿Has visto lo que ocurre? Has usado el método *close()* del objeto *window* para cerrar la ventana. Otro ejemplo escribe *javascript:window.alert(window.closed)*, ahora has usado el método *alert* del objeto *window* para mostrar una ventana con el valor de la propiedad *closed*, closed que dice si

la ventana está cerrada o abierta. Los objetos son bastante mas complicados que lo aquí expuesto, pero JavaScript no es un lenguaje orientado a objetos, aunque puede crearlos y por supuesto manipularlos. Los scripts manipulan los objetos propios del lenguaje y los que le proporciona el navegador. Y he aquí la causa de tus primeros dolores de cabeza como programador en javaScript: los sistemas de objetos de los diferentes navegadores no coinciden al 100%.

D) La ejecución de los scripts.

Habitualmente cuando quieres ejecutar un programa basta con buscar el archivo correspondiente y hacer un sencillo click de ratón o pulsar la tecla enter. ¿Pero que pasa con los programas en JavaScript?. En los ejemplos que has escrito en los apartados anteriores has enviado instrucciones en javascript al navegador, le has dado órdenes en directo. Pero esta no es la forma habitual de ejecutar programas en JavaScript. Lo normal es que la ejecución se realice de forma automática cuando el navegador carga una página, o cuando el usuario pasa el ratón por una imagen, o cuando pulsa el botón de un formulario, etc. Estos cambios provocan los llamados **eventos** que son recibidos por el navegador que reaccionará en la forma adecuada: si haces click en un hiperenlace se genera un evento y el navegador abre una nueva página. Esos eventos son los que se aprovechan para que se ejecuten las instrucciones que nosotros escribimos en JavaScript. A cada evento se le puede asociar una función para que haga algo predeterminado por nosotros. Por ejemplo cuando el navegador carga una página se produce un evento que puede aprovecharse para hacer que se abra otra ventana (las conocidas como ventanas popup tan usadas para mensajes publicitarios), o cuando pasamos el ratón por una enlace se produce otro evento que puede aprovecharse para llamar a una función que modifique el color en que se muestra el enlace, o cuando el usuario pulsa una tecla. Los eventos tienen la naturaleza de objetos, o sea, poseen métodos y propiedades. Así cuando se produce un evento podemos saber quien lo dispara, en que posición de la pantalla se ha disparado y otras propiedades dependientes de cada evento en concreto. Y aquí viene uno de las causas para tus futuros dolores de cabeza: cada navegador maneja los eventos de manera algo diferente.

Pero sigamos con el tema de la ejecución de los programas, veamos que es eso del **flujo de programa**. Cuando el navegador empieza a leer el script para ejecutarlo lo hace en orden secuencial, o sea, empieza con la primera instrucción, sigue por la segunda y así hasta llegar al final. Esto es lo que se conoce como ejecución secuencial o lineal. Pero a veces es necesario saltarse instrucciones, por ejemplo, construyes una página a la que sólo pueden entrar determinadas personas, deberás escribir una función que pida el nombre de quien desee ver la página, si es una persona autorizada muestra la página y si no lo es no la muestra. Tu programa no ha seguido un flujo lineal, unas veces ejecutará la parte de mostrar la página y otras no. Otra situación bastante común: deseas que tu programa recorra todas las imágenes de tu página y vaya cambiando su contenido, no vas a escribir el mismo código una y otra vez, lo ideal sería escribir las instrucciones y poderlas repetir.. Cualquier lenguaje de programación tiene solucionado este asunto mediante las llamadas sentencias de control del flujo de programa. Son sentencias que permiten que se ejecuten condicionalmente algunos pasos (condicionales) o repetir una serie de instrucciones una y otra vez (bucles). Estas instrucciones la veremos en capítulos posteriores, con ejemplos y probando cositas.

2) Elementos del Lenguaje:

A) Los datos

Anteriormente has hecho una primera aproximación a los elementos básicos con que vas a enfrentarte al programar. Es el momento de ahondar algo mas en estos elementos y comenzaremos por los más elementales: los datos. JavaScript maneja cuatro tipos de datos: numéricos, de cadena, booleanos y punteros. Los datos numéricos sirven para manejar cualquier número real, como 12, 56.5, -9. Pero también son datos numéricos: 2.0E2, 0xF0, 012. El primero corresponde a la llamada notación científica: $2.0*10^2$, mientras que el segundo ejemplo corresponde a la notación hexadecimal y el tercero a la octal. Lo normal es que sólo uses valores decimales y a veces hexadecimales.

Los datos de cadena son los usados para cadenas alfanuméricas, o sea, para representar palabras, así los siguientes son datos de cadena: *"prueba"*, *"La clave es 4r5t*". Observa que estos datos están encerados entre comillas, es la forma de decirle a JavaScript que está ante una cadena de caracteres y no ante un nombre o descriptor de algún elemento del programa. Es decir "41" es un dato de cadena mientras que 41 es el número cuarenta y uno. ¿Y que ocurre si quiero que la cadena tenga comillas?, por ejemplo queremos escribir la siguiente frase *Juan alias "Maqui" es el jefe,* en JavaScript esto es una cadena y debe encerrarse entre comillas "Juan alias "Maqui" es el jefe", la interpretación de esta cadena daría un error como puedes comprobar si escribes en la barra de direcciones la siguiente instrucción:

```
javascript:window.alert("Juan alias"Maqui" es el jefe")
```

¿Por qué? Pues por que ahí hay dos cadenas y un nombre en medio: "Juan alias", Maqui y "es el jefe". JavaScript intentará encontrar una variable con el nombre Maqui. Para poder hacer esto correctamente se usa el carácter de escape: \, lo que sigue a esta barra invertida es un carácter especial que debe interpretarse como un carácter normal, prueba ahora esto:

```
javascript:window.alert("Juan alias \"Maqui\" es el jefe")
```

Ahora ya funciona perfectamente. Existen mas caracteres especiales como tabulaciones, cambios de línea de escritura, borrado izquierda, la propia barra invertida. Puedes probarlos con la técnica del alert:

```
javascript:window.alert("Esto usa una \t tabulación ")
javascript:window.alert("Esto usa un \n cambio de línea")
javascript:window.alert("Esto usa un \r retorno de carro")
javascript:window.alert("Esto es la barra invertida \\ ")
```

Esto es bastante importante pues en muchas ocasiones usarás un programa para escribir segmentos de páginas HTML donde tendrás que usar caracteres especiales, como por ejemplo las comillas de los atributos.
Estos datos son bastante habituales incluso en la vida cotidiana, pero existen mas tipos de datos como seguro sabes. Por ejemplo si preguntas a alguien ¿son las cinco en punto? Sólo hay dos posibles respuestas: SI o NO. Este tipo de datos no es ni numérico ni de cadena, es **booleano**. En los lenguajes de programación en lugar de sí o no se

emplean los valores *true* o *false*. Estos datos son los que utilizaras cuando en los programas se tengan que tomar decisiones, o sea, comprobar algo y actuar en consecuencia: si el explorador es Netscape 4.0 no usar efectos de presentación, por ejemplo.

Existe un último tipo de datos que también se emplea a menudo en los scripts para la captura de eventos, son direcciones de memoria, o **punteros**, usadas para asignar funciones. Si en una variable se guarda el nombre de una función esa variable se convierte en otro nombre para esa función. ¿Raro? Esto tiene su verdadera utilidad al asignar funciones a los eventos disparados por el ratón.

B**)** Las variables.

Ya dijimos que los datos, los valores numéricos no siempre pueden utilizarse de manera directa, sino que habitualmente se almacenan en una posición de memoria con nombre, esto es lo que llamamos una **variable**. En otros lenguajes de programación existen diferentes tipos de variables en función del tipo de datos que pueden guardar: variables para cadenas, para números enteros, para números reales, etc. JavaScript es muy permisivo en este aspecto de manera que una variable puede guardar cualquier tipo de dato y además pueden crearse en cualquier parte del programa. Esto último es cómodo pero peligroso pues puede llevar a programas difíciles de depurar y de modificar, es conveniente llevar un cierto control sobre las variables que vas usando en cada función y declararlas al principio de la misma.

Hemos dicho que las variables tienen nombres, ¿cómo son estos nombres? Vale cualquier combinación de letras y dígitos, mas el guión bajo, siempre que el primer carácter no sea un dígito y por supuesto que no coincida con una palabra reservada del lenguaje, es decir, palabras con un significado especial para el intérprete como close, open, write... Es aconsejable usar nombres autoexplicativos, es una forma de documentar tus programas. Por ejemplo una variable para guardar una dirección de un icono puede llamarse direc_icono. Un último detalle JavaScript diferencia entre mayúsculas y minúsculas, así Edad y edad serían dos variables distintas.

Otro aspecto a tener en cuenta a la hora de usar las variables es su **ámbito**, es decir, qué funciones tienen acceso a ellas. Si se crea una variable dentro de una función sólo será conocida dentro de esa función, se trata de **variables locales**. Si se necesita que varias funciones tengan acceso a una determinada variable ésta debe crearse como **variable global**, esto se hace creándola fuera de todas las funciones. Por ejemplo en el siguiente script tenemos variables globales y locales:

```
<script language="Javascript">
var navegador_version = 0;
function verNavegador()
{
var version;
version = document.appVersion;
return version;
}
</script>
```

En este ejemplo navegador_version es una variable global mientras que version es local a la función verNavegador(). Observa que las variables están creadas con la palabra clave var, el uso de esta palabra es opcional, sólo es obligatorio si una variable local tienen el mismo nombre que una global. Otro detalle a tener en cuenta es que al mismo tiempo que creamos la variable podemos darle un valor, si no lo hacemos la variable contendrá el valor null.

C) Los Objetos

Javascript no posee todas las características de los lenguajes orientados a objetos como Java o C++, pero si es capaz de manejar objetos e incluso crearlos. De hecho si un programa en este lenguaje es capaz de interactuar con el explorador es gracias a esta capacidad. Javascript posee algunos objetos predefinidos u objetos intrínsecos como son: **Array**, **Boolean**, **Date**, **Function**, **Global**, **Math**, **Numbe**r, **Object**, **RegExp**, y **String**. Además el programador puede crear objetos nuevos, con sus propios métodos y propiedades, adaptados a las necesidades concretas de su aplicación.

Crear un objeto nuevo es tan simple como definir cuales serán sus propiedades y sus métodos, teniendo en cuenta que cualquier objeto que definamos ya posee heredados los métodos y propiedades del objeto predefinido **object**. En el ejemplo siguiente creamos un objeto **pagina** que aclarará la forma de crear objetos propios:

```
function pagina (titulo, color, fondo)
      {
        this.titulo = titulo;
        this.color = color;
        this.imgfondo = fondo;
        this.length = 3;
        }
var miPagina = new pagina("Mi página", "Blue", "cruces.gif");
var nuevapag = new pagina("2ª Página", "White", "");
```

Este objeto se crea con el operador **new** pasándole como argumentos las propiedades declaradas para este objeto: *titulo*, *color*, *imgfondo*. La palabra clave **this** se usa para referirnos al propio objeto que estamos definiendo. Aún mas podemos crear propiedades nuevas sólo para la variable **miPagina**, pero estas propiedades no afectarán al objeto **pagina** en sí. Por ejemplo:

```
miPagina.descripcion = "Este es un ejemplo";
alert (miPagina.descripcion);
```

da como resultado la frase *Este es un ejemplo*, pero si ahora escribiéramos:

```
alert(nuevaPag.descripcion);
```

obtendríamos **undefined** pues **descripcion** solo es una propiedad de la variable **miPagina** no del objeto **pagina**. Para ampliar un objeto usamos la propiedad prototype:

```
pagina.prototype.descripcion = "Objeto definido por mi";
alert(nuevaPag.descripcion);
alert(miPagina.descripcion);
```

Ahora hemos añadido una propiedad al objeto **pagina** y esta propiedad se transmite a las variables de tipo **pagina** (mas correctamente **instancias** de **pagina**), por tanto en ambos casos obtendríamos la frase *Objeto definido por mí*, que es el valor dado a la nueva propiedad.

D**)** Los Arrays

A pesar de su extraño nombre los **arrays** no son más que estructuras para almacenar listas de valores. A diferencia de otros lenguajes, en Javascript los arrays no son un tipo de variable sino que fueron implementados por Netscape como un objeto, lo cual les da una potencia enorme. Pero empecemos por abajo. Un array puede verse como una lista con nombre donde podemos anotar cosas, cada anotación se identifica por su número de orden en el array (la lista), su índice, y el número total de espacios para anotar cosas en el array es su longitud. Si en un array tenemos anotados 5 números, el primero de todos será el de índice 0 y el último tendrá como índice el 4, siendo 5 la longitud del array. Simple ¿verdad?, veamos como se crea un array y como se asignan valores en unos ejemplos muy sencillos:

```
....
semana = new Array(7);
miLista = new Array(1,5,9);
nombres= new Array('Juan', 'Luis', 'María');
vacio = new Array();
interesante = new Array(4);
...
```

El array semana se ha creado con longitud 7, o sea, permite 7 elementos. El array **miLista** se ha creado con longitud 3 y se ha dado valor a cada elemento al igual que el array **nombres**. Por último el array vacio se ha creado sin longitud.

```
...
semana[0] = 'Lunes';
semana[1] = 'Martes';
semana[2] = 'Miércoles';
semana[3] = 'Jueves' ;
semana[4] = 'Viernes';
semana[5] = 'Sábado';
semana[6] = 'Domingo' ;
vacio[5] = 'ultimo';
interesante['Jose'] = 10;
interesante['Pilar'] = 5;
interesante['Antonio'] = 8;
....
```

En este último segmento de código hemos rellenado los arrays: **semana** se ha rellenado con los nombres de los días de la semana. Observa el array **vacio**, hemos dado valor al elemento 5, a partir de ahora este array tiene longitud 6 con los cindo primeros elementos con valor null y el sexto (vacio[5]) con valor '*último*'. Por último el array llamado **interesante**, y lo es: los índices no tienen porque ser numéricos.

Cada elemento de un array puede ser un valor de cualquier tipo: números, cadenas ... u otro array, en este último caso tendremos arrays multidimensionales, o sea, donde cada elemento tiene varios índices o coordenadas para localizarlos. ¿Un lio? vemos el caso

mas simple, un array bidimensional sería un array donde cada elemento es a su vez otro array con lo que cada elemento tendrá dos índices uno para el primer array y otro para el segundo. El array bidimensional podmeos imaginarlo como una tabla ordenada en filas y columnas, cada elemento del array es una celda de esa tabla. Un ejemplo para una tabla de 3 filas y 2 columnas:

```
function matBidim()
{
var tabla = new Array(3);
tabla[0] = new Array(2);
    tabla[0][0] = 10;
    tabla[0][1] = 5;
tabla[1] = new Array(2);
    tabla[1][0] = 7;
     tabla[1][1] = 8;
tabla[2] = new Array(2);
     tabla[2][0] = 9;
     tabla[2][1] = 3;
alert(tabla[1][1]);                /*Visializaría un cuadro con el
número 8*/
}
```

Como ves la cosa es sencilla: el array **tabla** tiene tres elementos y cada uno de ellos es a su vez otro array de dos elementos.

E**)** Las Funciones

Las funciones como decíamos en la introducción, no son mas que bloques de instrucciones de programa con nombre y que pueden ejecutarse sin mas que llamarlas desde alguna parte de otra función o desde la página HTML, bien sea directamente o mediante eventos.

Habitualmente una función se crea para ejecutar una acción muy concreta. Javascript posee una serie de funciones predefinidas o **funciones globales** pero el programador puede crear las suyas propias. Para crear una función, tan sólo es necesario indicárselo al intérprete mediante la plabra clave **function** seguida del nombre que deseemos darle a la función y, encerrados entre paréntesis, las variables que simbolizan los valores con los que deba trabajar la función, los argumentos. Los paréntesis deben escribirse aunque no haya argumentos. Para los nombres de funciones seguimos las mismas reglas que para las variables: carateres, dígitos y guión bajo, debiendo comenzar por un carácter o el guión bajo.

Como es natural es recomendable dar a las funciones un nombre representativo de la operación que realize. Por supuesto no debemos darle un nombre que ya exista en JavaScript. A continuación encerradas entre llaves escribimos las sentencias u órdenes en JavaScript. Opcionalmente la función puede finalizar con la palabra clave **return** seguida de un valor, este valor será el que devuelva la función al programa que la llame. Por ejemplo:

```
function sumar(a,b)
{
var suma;
suma = a + b;
return suma;
}
```

Mediante este ejemplo creamos la función llamada **sumar**, que utiliza dos argumentos y lo que hace es sumar estos argumentos. Por último devuelve el resultado de la operación, mediante la palabra clave **return** seguida del valor que debe devolver. Ahora en el siguiente código de programa usamos la función recién definida:

```
var operacion;
operacion = sumar(4,5);
alert(operacion);
```

En este código llamamos a la función con los argumentos 4 y 5 y almacenamos el resultado en la variable **operacion**.
Hasta aquí el comportamiento de las funciones JavaScript es similar a cualquier otro lenguaje, pero en JavaScript las funciones también son objetos. Veamos el siguiente ejemplo:

```
var multip = new Function("x", "y", "return x * y")
alert(multip(8,9));
```

Interesante, ahora **multip** no es una variable cualquiera sino una instancia del objeto **Function** y puede usarse como la propia función. Esta característica permite asignar directamente funciones a los eventos de las páginas web y así simplificar su programación.

3) Los Operadores:

A) Operadores Aritméticos

En los primeros ejemplos de este tutor tan sólo se han usado sentencias muy simples como asignar un valor a una variable, mediante el **operador de asignación**, =, o realizar operaciones aritméticas, pero evidentemente JavaScript puede realizar mas operaciones. En esta seccion y las siguientes se presentan los operadores de que dispone este lenguaje clasificados en varios grupos, según el contexto en el que se usen. Comenzamos con los mas conocidos, los operadores aritméticos.

Suma +

Se trata de un operador usado para sumar dos valores numéricos o para concatenar cadenas entre sí o números y cadenas.

```
var var1 = 10, var2= "Buenos", var3 = " días", var4 = 31;
document.write(var1+var4)        /* resultado 41 */
document.write(var2+var3)       /* resultado: Buenos días */
document.write(var1+var3)       /* resultando: 10 días */
```

Resta -

Operador usado para restar valores numéricos. Puede actuar sobre un único operando numérico cambiándole de signo.

```
var num1 = 10, num2 = 8, res = 0;
res = num1 - num2;     /*res contiene 2 */
res = -res                        /* ahora res contiene -2*/
```

Producto (*) y cociente (/)

Realizan las operaciones aritméticas de multiplicar y dividir dos valores

```
var op1 = 50, op2= 4, div, mul;
div = op1/op2                /*div contiene 12.5 */
mul = op1 * op2          /*mul contendrá 200 */
```

Resto %

También llamado operador módulo calcula el resto de una división.

```
var op1 = 50, op2= 4, resto;
resto = op1 % op2;           /*resto contiene 2 */
```

Incremento (++) y decremento (--)

Estos operadores se usan para incrementar o decrementar en 1 el valor de una variable. Si el operador se antepone a la variable la operación de incremento o decremento es prioritaria sobre cualquier otra.

```
var op1=5, op2 = 5, res;
res = ++op1;          /*res adquiere el valor 6 y luego op1 el 6*/
res = op1++;          /*res adquiere el valor 5 y luego op2 el 6*/
```

Operadores compuestos

Los operadores +, -, *, / pueden asociarse con el operador de asignación (=) para cambiar el valor de una variable numérica por incrementándolo, decrementándolo, multiplicándolo o dividiéndolo por un valor. El operador += puede usarse igualmente con variables de cadena.

```
var num = 20, cad = "buena";
num += 5;           /*num adquiere el valor 25 (20 + 5) */
cad += 's'  ;         /*cad adquiere el valor 'buenas' */
num *= 10;         /*num adquiere el valor 250 (25*10) */
```

B) Operadores Binarios

E l ordenador, internamente, trata cualquier tipo de datos como una cadena binaria (ceros y unos). Así los números se representan en sistema binario de numeración mientras que los caracteres se convierten a código ASCII, que son números que se almacenan por tanto codificados en binario. JavaScript ofrece los operadores típicos para trabajar con estas cadenas a nivel de bit (cada uno de los ceros o unos de las cadenas binarias. Para trabajar con estos operadores es conveniente tener una idea previa sobre la codificación binaria.

Complementación ~

Complementa una cadena binaria convirtiendo los 1 en 0 y los 0 en 1.
Por ejemplo el número 38 escrito en sistema binario es 00100110 si le aplicamos este operador se convierte en 11011001, o sea el -39 (JavaScript usa codificación en complemento a 2 para los números negativos).

Desplazamiento izquierda <<

Desplaza los bits a la izquierda los lugares que se le indique rellenando con ceros por la derecha y desechando los bits de mayor peso, esto equivale a multiplicar por potencias de 2. Por ejemplo si al 00011010 (26) lo desplazamos 2 a la izquierda tendremos el 01101000 (104).

```
var num = 26, res;
res = num << 2;        /* num contendrá 104 */
```

Desplazamiento derecha >>

Desplaza los bits a la derecha los lugares que se le indique rellenando con ceros por la izquierda y desechando los bits de menor peso, esto equivale a una división entera por potencias de 2. Por ejemplo si al 00011010 (26) lo desplazamos 2 a la derecha tendremos el 00000110 (6).

```
var num = 26, res;
res = num << 2;        /* num contendrá 104 */
```

AND lógico binario &

Realiza un AND lógico bit a bit entre dos valores. El AND lógico da como resultado 1 sólo si ambos bits son 1. Por ejemplo

```
                  0 1 1 0 1 1 0 1 (109)
            AND 0 0 1 0 0 1 1 0 (38)
resultado:      0 0 1 0 0 1 0 0 (36)
```

```
var op1 = 109, op2 = 38, res;
res = op1 & op2;                    /*res contiene 36 */
```

OR lógico binario |

Realiza un OR lógico bit a bit entre dos valores. El OR lógico da como resultado 0 sólo si ambos bits son 0. Por ejemplo

```
                  0 0 1 1 1 0 1 0   (58)
            OR  0 1 0 1 0 0 1 0    (82)
resultado:      0 1 1 1 1 0 1 0   (122)
```

En el ejemplo podemos ver la sintaxis del operador

```
var op1 = 58, op2 = 82, res;
res = op1 | op2;                    /*res contiene 122 */
```

XOR lógico binario ^

Realiza un XOR lógico bit a bit entre dos valores. El XOR lógico da como resultado 1 si uno sólo de los bits es 1. Por ejemplo

```
                    0 0 1 1 1 0 1 0   (58)
              OR    0 1 0 1 0 0 1 0   (82)
resultado:          0 0 1 0 1 0 0 0   (40)
```

En el ejemplo podemos ver la sintaxis del operador

```
var op1 = 109, op2 = 38, res;
res = op1 ^ op2;                  /*res contiene 40*/
```

C) Operadores Lógicos

Loperadores lógicos se utilizan para realizar comparaciones entre valores, numéricos o no, dando como resultado un valor **booleanos** (**true**, **false**). La operación lógica negación invierte el operando, si es **true** lo hace **false** y viceversa. Si se comparan números con cadenas, JavaScript intenta convertir internamente los datos. En los operadores relacionales (>, <, >=, <=) intenta convertir los datos en tipo número. Para los operadores de igualdad (== !=) intenta convertir los tipos de datos a cadena, número y booleano. Los operadores de identidad (===, !==) no realizan conversión de tipo.

Mayor que >

Compara dos valores y devuelve true si el primero es mayor que el segundo. Compara tanto números como cadenas.

```
var hoy = 4; ayer = 10, comp;
comp = hoy > ayer          /* comp adquiere el valor false*/
```

Menor que <

Compara dos valores y devuelve true si el primero es mayor que el segundo. Compara tanto números como cadenas.

```
var hoy = 4; ayer = 10, comp;
comp = hoy < ayer          /* comp adquiere el valor false*/
```

Mayor o igual >=

Compara dos valores y devuelve true si el primero es mayor o es igual que el segundo. Compara tanto números como cadenas.

```
var hoy = 4; ayer = 4, comp;
comp = hoy >= ayer          /* comp adquiere el valor true*/
```

Menor o igual <=

Compara dos valores y devuelve true si el primero es menor o es igual que el segundo. Compara tanto números como cadenas.

```
var hoy = 4; ayer = 4, comp;
comp = hoy <= ayer          /* comp adquiere el valor true*/
```

Iguales ==

Compara dos valores y devuelve true si ambos son iguales. Compara tanto números como cadenas.

```
var hoy = 4; ayer = 4, comp;
comp = hoy == ayer          /* comp adquiere el valor true*/
```

Idénticos ===

Similar a == pero también compara el tipo de datos de los opeandos.
Compara dos valores y devuelve true si el primero es mayor o es igual que el segundo.

Compara tanto números como cadenas.

```
var hoy = 4; ayer = '4', comp;
comp = hoy == ayer;          /* comp adquiere el valor true*/
comp = hoy === ayer         /* comp adquiere el valor false*/
```

No iguales !=
No identicos !==
Invierten el sentido de las comparaciones iguales == e idénticos === respectivamente.

AND lógico &&
Este operador se utiliza para concatenar comparaciones, es decir, para comprobar varias condiciones. El resultado sólo será true si todas las comparaciones lo son.

```
var op1 = 2, op2 = 50, op3 = 25, comp;
comp = (op1 > op2) && (op1 < op3);       /*comp adquiere el valor
false */
```

comp es false por que op1 no es mayor que op2 aunque sea mayor que op3

OR lógico ||
Como el anterior, sirve apra realizar comparaciones compuestas y sólo devolverá false cuando todas las comparaciones los sean. Es decir basta que una comparación sea true para que devuelva el valor true.

```
var op1 = 2, op2 = 50, op3 = 25, comp;
comp = (op1 > op2) && (op1 < op3);       /*comp adquiere el valor
true */
```

comp es true por que op1 es menor que op3, (op1 < op3 es por tanto true)

D) Operadores Varios

delete
Se usa para borrar propiedades de un objeto o elementos de un array. Devuelve true si la operación se realizaó con éxito.

```
var lista = new Array(1,4,7,9,10);
delete(lista,0);
```

El elemento lista[1] contiene ahora undefined.

new
Se utiliza para crear instancias de un objeto

```
var hoy = new Date("10 /30/2000")
```

typeof
Devuelve el tipo de dato al que pertenece una variable o expresión. Los tipos devueltos son **number, string, boolean, object, function y undefined**.

```
hoy = 1.2345;
tipo = typeof(hoy);
```

La variable tipo contendrá **number**.

E) Funciones Globales

A sí como JavaScript proporciona objetos predefinidos, también posee una serie de funciones predefinidas. Se trata de las funciones: **eval, isNan, Number, String, parseInt, parseFloat, escape, unescape.**

eval
Se usa para evaluar una cadena con código JavaScript sin referirse a un objeto concreto. La sintáxis de **eval** es:
eval(expr)
donde expr es la cadena a evaluar.

isNaN(arg)
Determina si el argumento es un valor NaN (not a number)

parseInt(str, [base])
Convierte una cadena de caracteres en un valor numérico. La función lleva como argumento la cadena a convertir y opcionalmente puede llevar un segundo argumento para indicar la base de numeración en que está escrita la cadena. Si se omite se supone que la cadena representa un número en base 10. La cadena sólo podrá contener caractéres válidos para el sistema de numeración indicado: dígitos (0..9 para la base 10, 0 1 para números binarios, 0..7 para sistema octal, 0..9, A..F para sistema hexadecimal) y signo (+, -). Si encuentra algún carácter no válido sólo interpreta desde el principio de la cadena hasta el carácter no válido. Si comienza por un carácter ilegal devuelve **NaN**.
Ejemplo:

```
...
var minum1 = "14";
document.write(parseInt(minum1));
....
```

Escribirá 14. En el siguiente ejemplo transforma un número binario a decimal:

```
...
var minum1 = "11001";
document.write(parseInt(minum1,2));
....
```

Ahora escribirá 25, el equivalente decimal al binario 11001.

parseFloat(str)
Convierten una la cadena que se le pasa como argumento a un valor numérico de tipo flotante. Los caracteres válidos de la cadena son los mismos que en parseInt mas el punto decimal y el exponente (E). No admite un segundo argumento. Por lo demás funciona exactamente igual que **parseInt**.

```
...
var minum1 = "14.5E2";
document.write(parseInt(minum1));
....
```

Escribirá el número 1450, 14.5 por 10 elevado a 2.

Number(objArg) and String(objArg)
Permiten convertir el objeto pasado como argumento a un número o a una cadena. Por ejemplo:

```
...
var hoy = new Date();
hoy.getDate();
document.write(string(hoy));
....
```

Escribirá en pantalla la cadena "Sun Sep 3 20:40:05 UTC+0200 2000" si la fecha del día es domingo 3 de Septiembre y la hora es las 20:40:05.

escape(cadarg)
Codifica la cadena del argumento substituyendo todos los caracteres no ASCII por su código en el formato %xx. Por ejemplo:

```
....
var cadena = "Buenos días";
document.write(escape(cadena));
...
```

Produce la frase "Buenos d%EDas", pues la í (i acentuada) es el código hexadecimal ED de ese carácter.

unescape(cadarg)
Es inversa a la anterior, de manera que si la cadena contiene códigos del tipo %xx son convertidos al correspondiente carácter ASCII extendido.

```
....
var cadena = "Buenos d%EDas";
document.write(escape(cadena));
.....
```

Ahora se escribirá "Buenos días", se ha substituido %ED por su equivalente í (i acentuada).

F) Expresiones Regulares

Las expresiones regulares constituyen un mecanismo bastante potente para realizar manipulaciones de cadenas de texto. El proceso para el que se usan estas expresiones, presente en el mundo el UNIX y el lenguaje Perl, es el de buscar y/o sustituir una subcadena de texto dentro de otra cadena. En principio esto puede hacerse usando los métodos del objeto **string**, pero el problema surge cuando no tenemos una subcadena fija y concreta sino que queremos buscar un texto que responda a un cierto esquema, como por ejemplo: buscar aquellas palabras que comienzan con http: y finalizan con una \, o buscar palabras que contengan una serie de números consecutivos, etc.; es en estos casos cuando las expresiones regulares muestran toda su potencia. La subcadena que buscamos en el texto es lo que se llama un patrón y se construye encerrando entre dos barras inclinadas (/) una serie de caracteres normales y símbolos especiales llamados comodines o metacaracteres, (algo parecido a la orden dir *.bat usada en el DOS cuando queríamos listar los ficheros con extensión bat). Este patrón es una descripción del texto que se está buscando y JavaScript encontrará las subcadenas que concuerdan con ese patrón o definición. Las expresiones regulares se usan con el objeto **Regular Expresion** y también dentro de los métodos **String.match**, **String.replace**, **String.search** y **String.split**.

En la tabla que sigue se muestran los caracteres comodín usados para crear los patrones y su significado, junto a un pequeño ejemplo de su utilización.

	Significado	**Ejemplo**	**Resultado**
\	Marca de carácter especial	/\$ftp/	Busca la palabra $ftp
^	Comienzo de una línea	/^-/	Líneas que comienzan por -
$	Final de una línea	/s$/	Líneas que terminan por s
.	Cualquier carácter (menos salto de línea)	/\b.\b/	Palabras de una sola letra
|	Indica opciones	/(L|l|f|)ocal/	Busca Local, local, focal
( )	Agrupar caracteres	/(vocal)/	Busca vocal
[ ]	Conjunto de caracteres opcionales	/escrib[aoe]/	Vale escriba, escribo, escribe

La tabla que sigue describe los modificadores que se pueden usar con los caracteres que forman el patrón. Cada modificador actúa sobre el carácter o el paréntesis inmediatamente anterior.

	Descripción	**Ejemplo**	**Resultado**
*****	Repetir 0 o mas veces	/l*234/	Valen 234, 1234, 11234...
+	Repetir 1 o mas veces	/a*mar/	Valen amar, aamar, aaamar...
?	1 o 0 veces	/a?mar/	Valen amar, mar.
{n}	Exactamente n veces	/p{2}sado/	Vale ppsado
{n,}	Al menos n veces	/(m){2}ala/	Vale mmala, mmmala....
{m,n}	entre m y n veces	/tal{1,3}a/	Vale tala, talla, tallla

Los siguientes son caracteres especiales o metacaracteres para indicar caracteres de texto no imprimibles, como puedan ser el fín de línea o un tabulador, o grupos predefinidos de caracteres (alfabéticos, numéricos, etc...)

	Descripción	**Ejemplo**	**Resultado**
\b	Principio o final de palabra	/\bver\b/	Encuentra ver en *"ver de"*, pero no en *"verde"*
\B	Frontera entre no-palabras	/\Bver\B/	Empareja ver con *"Valverde"* pero no con *"verde"*
\d	Un dígito	/[A-Z]\d/	No falla en "A4"
\D	Alfabético (no dígito)	/[A-Z]\D/	Fallaría en "A4"
\O	Carácter nulo		
\t	Caracter ASCII 9 (tabulador)		
\f	Salto de página		
\n	Salto de línea		
\w	Cualquier alfanumérico, [a-zA-Z0-9_]	/\w+/	Encuentra *frase* en "*frase*.", pero no el . (punto).
\W	Opuesto a \w ([^a-zA-Z0-9_])	/\W/	Hallaría sólo el punto (.)
\s	Carácter tipo espacio (como tab)	/\sSi\s/	Encuentra *Si* en *"Digo Si "*, pero no en *"Digo Sientate"*
\S	Opuesto a \s		
\cX	Carácter de control X	\c9	El tabulador
\oNN	Carácter octal NN		
\xhh	El hexadecimal hh	/\x41/	Encuentra la *A* (ASCII Hex41) en *"letra A"*

4) La gramática:

A) Condicionales

El orden en que se ejecutan las instrucciones de un programa es, por defecto, secuencial: ejecuación instrucción tras instrucción. Así un programa se escribirá como una sucesión de instrucciones o sentencias, utilizando un punto y coma para indicar el final de la instrucción. Pueden agruparse una serie de sentencias en un bloque encerrándolas entre llaves. A veces es necesario alterar este orden para ello se utilizan las instrucciones de control: condicionales, selección y bucles. Serán las sentencias condicionales las primeras que veremos.
Una sentencia condicional es una instrucción en la que se hace una comparación y según el resultado verdadero o falso (**true** o **false**) de la misma el programa seguirá ejecutando una u otra instrucciones. La condiconal mas simple que podemos escribir es aquella que ejecuta u omite una serie de sentencias dependiendo de si la comprobación da verdadero o falso. La sintaxis de esta sentencia es.

if (condición)
    {bloque a ejecutar si la condición es cierta}
else
    {bloque a ejecutar si la condición es false}

Si omitimos la parte del **else** tendremos una condicional simple. Esta sintaxis en algunos casos puede simplificarse utilizando la siguiente forma:

(condición) **?**{bloque si cierta} **:** {bloque si falsa}

En el siguiente ejemplo evitamos realizar una división por cero

```
.......
if (div == 0)
      alert('No se puede dividir por 0');
else
      coc = num / div;
.........
```

Otro ejemplo usando la segunda forma:

```
............
cad = (num >= 0) ? ' + ' : ' - ';
........
```

En este ejemplo **cad** tomará el valor **+** si num es positivo o cero y el **-** si es negativo.
Las sentencias **if** pueden anidarse, es decir, dentro de una sentencia if pueden meterse mas sentencias **if**.
Las condiciones pueden ser sencillas como en estos ejemplos o pueden enlazarse usando los operadores **&&** y || (AND y OR lógicos). Veamos un ejemplo en el que comprobamos si un número está comprendido entre 1 y 5:

```
if ((num>=1) y (num < 5)
      {
        lista[indice] = 'Muy bajo';
        bajos++;
      }
indice++;
```

En este ejemplo si **num** está entre 1 y 5 (excluido) se anota en una lista la palabra 'Muy bajo' y se incrementa la variable **bajos.** Como vemos no se ha usado la parte de **else** y como se deben ejecutar mas de una sentencia las hemos encerrado entre llaves. Si **num** no cumple la condición el programa se salta este bloque. En cualquier caso la siguiente instrucción que se ejecute tras el condicional será la que incrementa el valor de **indice**.

B) Selección Múltiple

L a estructura condicional permitía a lo sumo elegir entre dos posibles caminos en la ejecución de un programa: si la condición era cierta se ejecuta un bloque y si no se ejecuta otro. Pero pueden existir casos en los que el programa deba tener mas de dos alternativas, por ejemplo si queremos un programa que presente un título en un idioma de cuatro posibles. Esto puede solucionarse mediante varios **if** anidados, siguiendo el ejemplo tenemos que elegir entre idiomas: castellano, ingles, francés y alemán.

```
      if (leng == "castellano")
     pagCast();
else
     if (leng == "ingles")
         pagIng();
     else
          if (leng == "frances")
              pagFran();
          else
                 if (leng == "alemán")
                     pagAlemt();
                 else
                     error('idioma no presente');
```

Como vemos resulta un código bastante complejo. Para estos casos disponemos de la sentencia switch...case...default, de selección múltiple. El ejemplo anterior quedaría:

```
... ... ...
switch (idioma) {
      case 'castellano' :
            pagCast();
            break;
      case 'ingles' :
            pagIng();
            break;
      case 'frances' :
            pagFran();
            break;
      case 'alemán' :
            pagAlem();
            break;
      default :
            error ('Idioma no presente');
}
```

Durante la ejecución se compara la variable **idioma** con cada uno de los posibles valores y cuando coincidan ejecuta el código correspondiente. La instrucción **break** pone fin al bloque y hace que el programa salte a la instrucción siguiente a la sentencia switch(), si se omite el programa continuaría con la siguiente comparación. La sección del **default** es opcional, su finalidad es ejecutar algún código cuando ninguna de las condiciones se cumpla.

C) Bucles

A veces es necesario repetir un mismo conjunto de sentencias varias veces. Por ejemplo para borrar todos los elementos de un array simplemente debemos hacer delete en cada uno de ellos, es una sentencia que se repetirá tantas veces como largo sea el array. Este es un típico trabajo para las estructuras repetitivas o bucles. En esencia la ejecución de un bucle consiste en ejecutar repetidas veces una misma parte del programa (cuerpo del bucle) hasta que se cumpla una determinada condición, en cuyo caso se acaba la repetición y el programa continúa con su flujo normal. Existen varias sentencias de bucles: while (condición) {... } , do {...} until (condicion) y for(contador; condición; modcont){...}.

Sentencia while

En esta estructura el programa primero comprueba la condición: si es cierta pasa a ejecutar el cuerpo del bucle, y si es falsa pasa a la instrucción siguiente a la sentencia while. Como siempre un ejemplo lo aclarará todo:

```
var lista = new Array(10);
var ind=0;
while (ind < 10)
     {
       lista[ind] = '0';
       ind++;
      }
```

En este ejemplo mientras que el valor almacenado en **ind** sea menor que 10 (la longitud del **array**) irá almacenando en cada elemento del array lista un 0 e incrementando el valor de **ind**. Cuando este valor sea 10 el programa no entrará en el cuerpo del bucle. Si no se incrementara el valor de **ind** el bucle no acabaría nunca, el programa quedaría ejecutando indefinidamente el cuerpo del bucle.

Sentencia do...while

Se trata de un bucle en el que la condición se comprueba tras la primera iteración, es decir que el cuerpo del bucle se ejecuta al menos una vez. El ejemplo anterior quedaría como sigue

```
var lista = new Array(10);
var ind=0;
do
       lista[ind] = '0';
       ind++;
while (ind < 10)
```

Como vemos aquí no son imprescindibles las llaves para encerar el cuerpo del bucle. No está contemplada en el standard ECMA 1.5.

Sentencia for

Esta sentencia utiliza una variable de control a modo de contador para controlar la repetición del cuerpo del bucle. La sentencia da un valor inicial a este contador y en cada iteración lo modifica según le indiquemos y comprueba la condición, si se cumple ejecuta el cuerpo del bucle, si no lo salta y continúa por la siguiente sentencia. Vemos el ejemplo anterior usando esta sentencia:

```
var lista = new Array(10);
var ind;
for (ind=0; ind < 10; ind++)
```

```
      {
        lista[ind] = '0';
       }
```

Como vemos el cuerpo del bucle no incrementa la variable **ind**, esto se indica en la cabecera de la sentencia. Este código hace exactamente lo mismo que el anterior.

Sentencia for ... in

Se trata de una variante de la sentencia **for** utilizada para iterar o recorrer todos los elementos de un objeto o de un array. Usa una variable de control que en cada iteración toma el valor del elemento del objeto recorrido. Por ejemplo si pruebas este código podrás ver todos los elementos del objeto document

```
var item;
for (item in document)
         document.write(item+'<br>');
```

Con una matriz la variable de control toma el valor de los índices de la matriz, no su contenido.

D) Ruptura de Bucles

Aunque procuremos usara una programación estructura alguna vez puede ser necesario interrumpir la repetición de un bucle o forzar una iteración del mismo, esto puede lograrse mediante las sentencias **break** y **continue**. Son sentencias aplicables a cualquiera de las estructuras de bucle en JavaScript.

break

La sentencia break interrumpe la iteración actual y envía al programa a la instrucción que sigue al bucle.

```
var lista = new Array ('a','b','c','z','x','f');
var item ;
for (item in lista)
         {
            if (lista[item] == "z")
                        break;
             document.write(lista[item]+'<br>');
}
```

Este ejemplo escribiría el contenido del array **lista** hasta encontrar una letra **z**.

continue

La sentencia **continue** interrumpe la iteración actual y envía al programa a la comprobación de la condición, si esta es cierta continúa con la siguiente iteración.

```
var lista = new Array ('a','b','c','z','x','f');
var item ;
for (item in lista)
         {
            if (lista[item] == "z")
                        continue;
             document.write(lista[item]+'<br>');
}
```

Este ejemplo escribiría el contenido del array saltándose la letra **z.**

5) Los Objetos:

A) Arrays

Como objetos que son, los **arrays** poseen sus **propiedades** y **métodos** predefinidos, que son ampliables por el usuario. Es necesario hacer notar que estos métodos y propiedades son los definidos para el JavaScript 3.0 de Microsoft. Netscape añade mas métodos en su versión, pero los aquí definidos son comunes a ambos navegadores.

Propiedades

length
Como su nombre indica esta propiedad nos devuelve la longitud del array, es decir, el número de elementos que puede almacenar. Su uso es muy simple:

```
      var lista = new Array(50);
tamagno = lista.length;        /*tamagno almacenaría el valor 50 */
```

prototype
Esta es una propiedad muy potente en el sentido que nos permite agregar al objeto Array las propiedades y métodos que queramos.

```
      Array.protoype.descriptor = null;
dias = new Array ('lunes', 'Martes', 'Miercoles', 'Jueves',
'Viernes');
dias.descriptor = "Dias laborables de la semana";
```

En este ejemplo hemos creado una nueva propiedad para el objeto array, la propiedad descriptor que podría utilizarse para darle un título a la matriz.

Métodos

concat(objArray)
Une el objeto Array con el array que se le pasa como argumento y devuelve el resultado en un nuevo array, sin modificar los arrays que se concatenan.

join()
Convierte los elementos de un array en una cadena separados por el carácter que se le indique. El separador por defecto es la coma.

```
      a= new Array("Hola","Buenos","días");
document.write(a.join() +" <br>");
document.write(a.join(", ") +" <br>");
document.write(a.join(" + ") +" <br>") ;
```

La salida de este programa sería
Hola,Buenos,Días
Hola, Buenos, Días
Hola+Buenos+Días

reverse()
Invierte el orden de los elementos de un Array en el propio array, sin crear uno nuevo.

slice(ini, fin)
Extrae parte de un Array devolviéndolo en un nuevo objeto Array.

```
      lista = new Array('a', 'b', 'c', 'd', 'e', 'f', 'g', 'h');
sublista = lista.slice(2,6);
alert(sublista.join());
```
En el ejemplo sublista contendrá los elementos desde el índice 2 al 5 ambos inclusive, o sea, 'c', 'd', 'e', 'f'. Si se omite el segundo argumento se extrae hasta el último elemento del array y si es negativo se entiende como contando desde el final.

sort(rutord)
Ordena alfabéticamente los elementos de un objeto Array. Opcionalmente podemos pasar como argumento una función para determinar el orden, esta función posee dos argumentos y devolverá un valor negativo si el primer argumento es menor que el segundo, cero si son iguales y un valor positivo si el primer argumento es mayor que el segundo. En castellano esto es necesario si queremos que la ñ y vocales acentuadas figuren en su lugar.

B) Booleanos: Verdadero o Falso

Las variables booleanas o lógicas son las que sólo pueden tomar dos valores: **true**, verdadero, y **false**, falso. Este tipo de variables está implementado en JavaScript como un objeto.

Métodos
toString
Si el valor es false devuelve la cadena "false" y si es true devuelve la cadena "true"
valueOf
Devuelve el valor booleano (**true** o **false**)

Propiedades
constructor
heredada del objeto genérico **Object**, devuelve la referencia al constructor:
function Boolean() { [native code] }
prototype
Es una propiedad utilizada para asignar nuevos métodos o propiedades, heredado del objeto genérico Object. Por ejemplo podemos traducir los valores **true** o **false**.

```
      function valor () { return this.valueOf()?'cierto':'falso' }
Boolean.prototype.valor = valor;
var item = new Boolean(false);
document.write(item.valor());
```
Con este ejemplo logramos que **true** se devuelva como la cadena "cierto" y **false** como la cadena "falso".

C) Objeto Function

Permite la creación de **funciones**, ya sean con nombre o anónimas. La creación de una función puede realizarse por el método tradicional y común a la mayoría de lenguajes de programación:

```
function sumar(a, b)
{
return a+b;
}
```

O bien mediane el conocido operador **new**:

```
sumar = new Function ("a", "b", "return a+b");
```

En cualquier caso la función se usará de igual forma:

```
document.write( sumar(90, 100) );
```

Métodos

Los heredados del objeto **Object**

Propiedades

arguments

Se trata de un **array** que contiene los argumentos pasados a la función. Esta propiedad permite el uso de funciones con un número variable de argumentos.

caller

Contiene una referencia a la función que llamó a la actual.

constructor

Heredada de la clase **Object**

D) Objeto Number

Es el objeto destinado al manejo de datos y constantes numéricas. Realmente no es habitual crear objetos de este tipo por cuanto JavaScript los crea automáticamente cuando es necesario. No obstante la sintaxis para su creación es la habitual para cualquier objeto:
minúmero = new Number(valorinicial)
El valor inicial es optativo, si no se usa el objeto se crea con valor null

Métodos

L os heredados del objeto **Object**

Propiedades

Además de las heredadas del objeto **Object** Number posee las siguientes propiedades:

MAX_VALUE

Indica el valor máximo utilizable por JavaScript, actualmente 1.79E+308.

MIN_VALUE

Indica el valor mínimo utilizable por JavaScript, actualmente 2.22E-308.

NaN

Una constante usada para indicar que una expresión ha devuelto un valor no numérico. NaN no puede compararse usando los operadores lógicos habituales, para ver si un valor es igual a NaN se debe usar la función incorporada **isNaN**

NEGATIVE_INFINITY

Una constante para indicar infinito positivo, es decir, un valor superior al MAX_VALUE

POSITIVE_INFINITY

Una constante para indicar infinito negativo, es decir, un valor superior al MAX_VALUE con signo negativo

E) Objeto Object

Pues sí: existe un objeto llamado **Object** del que derivan todos los objetos de JavaScript, los predefinidos y los definidos por el usuario. Esto significa que los objetos usados en JavaScript heredan las propiedades y métodos de **Object**.

Métodos

toString

Devuelve una cadena dependiendo del objeto en que se use

Objeto	**Cadena devuelta por el método**
Array	Los elementos del array separados por coma
Boolean	Si el valor es false devuelve "false" si no devuelve "true"
Function	La cadena "function nombre_de_función(argumentos){ [código]}"
Number	Representación textual del número
String	El valor de la cadena
Default	"[object nombre_del_objeto]"

valueOf

Devuelve el valor del objeto dependiendo del objeto en que se use

Objeto	**Valor que devuelve el método**
Array	Una cadena formada por los elementos separados por coma
Boolean	El valor booleano (true o false)
Date	La fecha como el número de milisegundos desde el 1/1/1970, 00:00
Function	La propia función
Number	El valor numérico
String	La cadena
Default	El propio objeto

Propiedades

constructor

Esta propiedad contiene una referencia a la función que crea las instancias del objeto en particular. Por ejemplo:

```
      x = new String("Hola");
//En este caso s.constructor contendrá
//              function String() { [native code] }
```

prototype

Es una propiedad utilizada para asignar nuevos métodos o propiedades a un objeto, elementos estos que serán heredados por las diferentes instancias de ese objeto. Ejemplo:

```
      Array.prototype.nombTipo = "matriz";
lista = new Array(9);
document.write(lista.nombTipo);
//Escribirá la palabra matriz que es el nombTipo
//que hemos dado para el objeto Array
```

F) Objeto Regular Expession

JavaScript usa este objeto para trabajar con patrones de expresiones regulares, estos patrones se crean como cualquier otro objeto mediante su inicialización directo o bien mediante el constructor **new RegExp(),** como podemos ver en el ejemplo:

```
......
var mipatron = /^[aeiou]/gi
var mipatron2 = new RegExp("^[aeiou]", "gi")
........
```

Ambas formas conducen al mismo patrón, en este ejemplo define palabras que comienzan con una vocal. El patrón puede llevar modificadores o flags para matizar la forma de buscar, en el ejemplo se usan dos: g i Estos modificadores tienen los siguientes sginficados:

flags	**Significado**
g	Explorar la cadena completa
i	No distinguir mayúsculas de minúsculas
m	Permite usar varios ^y $ en el patrón
s	Incluye el salto de línea en el comodín punto **.**
x	Ignora los espacios en el patrón

Estos patrones poseen en total tres métodos **exec()**, **test()**, **compile()** además de los métodos ya citados del objeto String que también usan patrones como son: match(), replace(), search() y split(). La única propiedad que funciona en estos objetos es la source que refleja el contenido del patrón. En el patrón pueden aparecer caracteres o grupos de caracteres encerrados entre paréntesis, posteriormente podemos usar un índice para referirnos individualmente al contenido de esos paréntesis.
Por ejemplo vamos a sustituir por un - todas los dígitos situados tras una letra en la cadena a explorar.

```
      var cadexp = "234879x089h9y7";
var patron = /([a-z])(\d)/ig;
document.write(cadexp+'<br> ');
cadexp = cadexp.replace(patron, "$2-");
document.write(cadexp)
```

Como ves donde antes existía un dígito seguido de una letra ahora hay un dígito seguido de un guión. Las coincidencias con el primer paréntesis del patrón están en $1 y con el segundo en $2. La primera coincidencia hallada es x0, luego $1 contiene **x** y $2 contiene **0**, sustituyo lo hallado con -$2, o sea, quito $1 y pongo un guión y me quedará **-0** en lugar de **x0**. Como se ha usado el flag **g** (global) esta operación se realiza en toda la cadena.

a) Métodos RegExp: Compile (cadpatr)

Un patrón de búsqueda puede construirse mediante una simple asignación o mediante el constructor new RegExp y ser utilizada tal cual, pero se puede mejorar bastante la búsqueda usando este método que convierte el patrón en un formato interno para optimizar su uso. Utiliza como argumento una cadena que representa la expresión regular que se quiere compilar

```
var patron = new RegExp();
patron.compile("\\D-");
var busq = patron.exec("1234u90t-789");
document.write('Buscando '+patron.source+'<br>');
document.write(busq[0]+' está en la posición ' +
               busq.index +' de busq.input');
```

En este ejemplo se busca cualquier no numérico seguido de un guión en la cadena "1234u90t-789". Primero se declara la variable patron y se compila con el patrón \D- que indica cualquier carácter no numérico seguido de guión. Por último muestra el patrón usado y los resultados de la búsqueda: coincidencia encontrada, posición y cadena explorada.

b) Métodos RegExp: Exec (cadexplor)

Este método busca la primera concordancia del patrón con el contenido de la cadena de texto donde se busca, que se le pasa como argumento. Si no encuentra ninguna concordancia devuelve null, pero encuentra una secuencia de caracteres que se adapte al patrón de búsqueda devuelve un array cuyo primer elemento indica la concordancia encontrada y las restantes indican los resultados de acuerdo a los paréntesis que aparezcan en la expresión regular. Además este array posee dos propiedades**: index**, para indicar la posición de la subcadena encontrada, y **input**, que contiene la cadena de caracteres que se está explorando. Además modifica las propiedades de una variable global RegExp con datos relativos a la búsqueda. En el ejemplo que sigue buscamos cualquier letra seguida de un número y de un guión, el patrón de búsqueda será /[a..z]\d-/i, [a..z] representa todas las letras del abecedario, \d representa cualquier número y el modificador i se usa para no diferenciar mayúsculas de minúsculas.

```
patron = /[a..z]D\d-/i;
var busca = new Array()
busca = patron.exec("3c491a-9d1d6-91br");
if (busca != null){
   document.write("Concuerda en: "+busca.index + '<br>');
   document.write("Explorando:" +busca.input + '<br>');
   document.write("Hallado: " + busca[0] + '<br>');
}
document.write("Resto " + RegExp.rightContext + '<br>');
```

c) Métodos RegExp: Test (cadexp)

Este es el método más simple del objeto expresión regular, tan sólo comprueba si existe alguna coincidencia en la cadena explorada, pasada como argumento, con el patrón de búsqueda. Si existe tal coincidencia devuelve un valor booleano true y en caso contrario devuelve false. Además afecta a las propiedades del objeto global RegExp.

```
var patron = new RegExp("Lunes","gi");
var cadexpl = "La reunión es el lunes o el martes.";
var eslunes = patron.test(cadexpl);
document.write("¿Es el lunes? "+eslunes+'<br>');
document.write("Hallado en "+RegExp.index);
```

En este sencillo ejemplo se comprueba si la cadena explorada, **cadexpl**, contiene la palabra "lunes", sin considerar la caja (mayúsculas o minúsculas). El resultado lo guarda en la variable **eslunes.**

c) RegExp

Se trata de una variable global usada por JavaScript cuando realiza operaciones donde intervengan expresiones regulares. Cada vez que se realiza una de estas operaciones se modifican las propiedades de esta variable. Es una variable que puede consultarse pero sobre la que se puede modificar directamente, es de sólo lectura. No tiene ningún

método asociado y sus propiedades siempre hacen referencia a una operación de búsqueda, sea con los métodos de un objeto Regular Expresion o del objeto string.

Propiedades

$1..$9:
Estos índices contienen las partes agrupadas con paréntesis en el patrón de búsqueda.

input
Cadena que se ha explorado.

lastmatch
Última coincidencia encontrada.

multiline
Variable booleana que indica si la cadena explorada incluye saltos de línea.

lastParen
Última coincidencia encontrada con un patrón entre paréntesis.

leftContext
Toda la cadena hasta la coincidencia hallada.

rightContext
Toda la cadena desde la coincidencia hasta el final de la cadena.

Estas propiedades sólo son de lectura y son actualizadas cada vez que se realiza alguna búsqueda con patrón, sea con los métodos de una expresión regular o con los de String. En el siguiente ejemplo se puede observar estos valores tras una operación de búsqueda.

```
var patron= /\D(\d)(\D)/g;
var buscaren = "abde5fghj45oi";
var hallado = patron.exec(buscaren);
var item; document.write("$1: "+RegExp.$1+"<br>");
document.write("$2: "+RegExp.$2+"<br> ");
document.write("input: "+RegExp.input+"<br> ");
document.write("index: "+RegExp.index+"<br> ");
document.write("lastIndex: "+RegExp.lastIndex+"<br> ") ;
document.write("multiline: "+RegExp.multiline+"<br>");
document.write("lastMatch: "+RegExp.lastMatch+"<br>");
document.write("lastParen: "+RegExp.lastParen +"<br>");
document.write("leftContext: "+RegExp.leftContext +"<br>");
document.write("rightContext:"+RegExp.rightContext+"<br>");
```

Si pruebas este ejemplo con MSIExplorer y Netscape podrás ver que no todas estas propiedades son reconocidas por ambos navegadores: Netscape y MSIexplorer.

G) Objeto String

El objeto **String** se usa para manipular cadenas de caracteres. En JavaScript todo texto encerrado entre comillas, dobles o simples, se interpreta como una cadena, así '45' no es el número cuarenta y cinco sino la cadena formada por los caracteres 4 y 5. El objeto **String** permite realizar operaciones con cadenas como concatenar o dividir cadenas, buscar texto, extraer parte de un texto, etc.. La operación de crear una variable de este tipo se lleva a cabo como es habitual con el operador **new** pudiéndole pasar un argumento para inicializar la variable. Al usar un método o referirnos a una propiedad podemos usar el nombre de la variable o una constante de cadena así el ejemplo

```
var mitexto = "Esta es una cadena";
var pos = mitexto.indexOf("una")
```

puede también escribirse en la siguiente forma:

```
var pos = "Esta es una cadena". indexOf("una");
```

Métodos

anchor	**fromCharCode**	**small**
big	**indexOf**	**split**
blink	**italics**	**strike**
bold	**lastindexOf**	**sub**
charAt	**link**	**substr**
charCodeAt	**match**	**substring**
concat	**replace**	**sup**
fixed	**search**	**toLowerCase**
fontcolor	**slice**	**toUpperCase**
fontsize		

Propiedades
length: devuelve la longitud de la cadena.
prototype: permite agregar métodos y propiedades al objeto

a) Métodos de String: anchor(atrcad)

Este método crea, a partir de un objeto String, una cadena conteniendo un elemento HTML ANCHOR, con el atributo NAME igual a la cadena que se le pase en **atrcad**.

```
var refer = "Referencia num. 1" ;
var ancla = refer.anchor("Refer1");
```

El valor de la variable **ancla** será:

```
<A NAME="Refer1">Referencia num. 1</a>
```

La sintaxis de este método permite usar una constante String en lugar del nombre de un objeto String. El ejemplo anterior podría haber escrito como:

```
var ancla = "Referencia num. 1".anchor("Refer1");
```

b) Métodos de String: big()

Este método devuelve una cadena consistente en el objeto String rodeado con las etiquetas <BIG> </BIG> del lenguaje HTML. Por ejemplo:

```
var mitexto = "Este es el texto";
mitexto = mitexto.big();
```

Tras la última sentencia la variable **mitext** contendrá

```
<big>Este es el texto</big>
```

Se puede usar una constante de cadena en lugar de un nombre de variable, así el ejemplo podría haberse escrito:

```
var mitexto = "Este es el texto".big();
```

c) Métodos de String: blink()

Este método devuelve una cadena consistente en el String rodeado con las etiquetas <blink></blink> del lenguaje HTML. Por ejemplo:

```
var mitexto = "Texto para intermitente";
mitexto = mitexto.blink();
```

Tras la última sentencia la variable mi texto contendrá el valor:

```
<blink>Texto para intermitente</blink>
```

d) Métodos de String: bold()

Este método devuelve una cadena consistente en el String rodeado con las etiquetas <B> </B>, negrita, del lenguaje HTML. Por ejemplo:

```
var mitexto = "Texto para negrita";
mitexto = mitexto.bold();
```

Tras la última sentencia la variable mi texto contendrá el valor:

```
<B>Texto para negrita</B>
```

e) Métodos de String: charAt(atrent)

Este método aplicado a una cadena devuelve el carácter que se encuentra en la posición dada por el atributo **atrent**, teniendo en cuenta que el índice del primer carácter a la izquierda de la cadena es 0 y el último es una unidad menor que longitud de la cadena. Si el valor del atributo no es válido (igual o mayor que la longitud de la cadena o negativo) el método devuelve el valor **undefined**. Por ejemplo el siguiente código devuelve la posición del tercer carácter de la cadena **nombre**:

```
var nombre = "abcdefghij";
var car3 = nombre.charAt(2);
```

Devolverá "c", que es el tercer carácter por la izquierda (índice igual a 2).

f) Métodos de String: charAt(atrent)

Este método aplicado a una cadena devuelve el código Unicode del carácter que se encuentra en la posición dada por el atributo **atrent,** teniendo en cuenta que el índice del primer carácter a la izquierda de la cadena es 0 y el último es una unidad menor que longitud de la cadena. Si el valor del atributo no es válido (igual o mayor que la longitud de la cadena o negativo) el método devuelve el valor **NAN**. Por ejemplo el siguiente código devuelve el Unicode del tercer carácter de la cadena **nombre**:

```
var nombre = "abcdefghij";
var car3 = nombre.charAt(2);
```

Devolverá 99, que es el código de la letra 'c', el tercer carácter por la izquierda (índice igual a 2).

g) Métodos de String: concat(atrcad)

Este método aplicado a una cadena le agrega la cadena pasada como atributo, **atrcad,** que será una variable o constante literal, cualquier otro tipo es convertido a cadena. Por ejemplo el siguiente código concatena 'Buenos ' y 'días':

```
var saludo = "Buenos ";
var hola = saludo.concat("días");
```

La variable **hola** contendrá "Buenos días", es lo mismo que si se hubiera escrito:

```
var hola = saludo + "días"
```

h) Métodos de String: fixed()

Este método devuelve una cadena consistente en el objeto String rodeado con las etiquetas <TT> </TT>, espaciado fijo o teletype, del lenguaje HTML. Por ejemplo:

```
var mitexto = "Este es el texto";
mitexto = mitexto.fixed();
```

Tras la última sentencia la variable mitext contendrá

```
<TT>Este es el texto</TT>
```

Se puede usar una constante de cadena en lugar de un nombre de variable, así el ejemplo podría haberse escrito:

```
var mitexto = "Este es el texto".fixed();
```

i) Métodos de String: fontcolor(atrcad)

Este método crea, a partir de un objeto String, una cadena conteniendo un elemento FONT del lenguaje HTML con el atributo COLOR igual a la cadena que se le pase en **atrcad**.

```
var mitexto = "Texto en color" ;
mitexto = mitexto.fontcolor("#FFAC3E");
```

El valor de la variable **ancla** será:

```
<FONT COLOR="#FFAC3E">Texto en color</FONT>
```

La sintaxis de este método permite usar una constante String en lugar del nombre de un objeto String. El ejemplo anterior podría haber escrito como:

```
var mitexto = "Texto en color".fontcolor("#FFAC3E");
```

j) Métodos de String: fontsize(atrnum)

Este método crea, a partir de un objeto String, una cadena conteniendo un elemento FONT del lenguaje HTML con el atributo SIZE igual al valor entero que se le pase en **atrnum**.

```
var mitexto = "Texto de prueba" ;
mitexto = mitexto.fontsize(-1);
```

El valor de la variable **mitexto** será:

```
<FONT SIZE="-1">Texto de prueba</FONT>
```

La sintaxis de este método permite usar una constante String en lugar del nombre de un objeto String. El ejemplo anterior podría haber escrito como:

```
var mitexto = "Texto de prueba".fontsize(-1);
```

k) Métodos de String: fromCharCode(cod1, cod2, ...)

Este es un método global del objeto String que crea una cadena a partir de los códigos Unicode que se le pasen como parámetros. Por ejemplo:

```
var cadena = String.fromCharCode(65,66,67);
```

La variable cadena contendrá "ABC", que son los caracteres con los códigos 65, 66 y 67.

l) Métodos de String: indexOf(atrcad, desde)

Este método devuelve la primera posición dentro del objeto String donde comienza la subcadena pasada como argumento en **atrcad**. Admite un segundo argumento opcional que indica desde que posición debe realizar la búsqueda, si se omite comienza a buscar por el primer carácter de la izquierda. Valores del segundo argumento negativos o mayores que la longitud de la cadena se consideran 0. Si la subcadena no se encuentra el valor devuelto es -1. Por ejemplo:

```
var cadena = "mi.correo@mail.com";
var arroba = cadena.indexOf('@');
var punto = cadena.indexOf('.',arroba);
```

Este ejemplo devuelve en **arroba** la posición 9 mientras que **punto** contiene la 14 pues la búsqueda se hizo desde la posición donde está el carácter arroba y encuentra el segundo punto. Recuerda que las posiciones en las cadenas se cuentan desde 0.

m) Métodos de String: italics()

Este método devuelve una cadena consistente en el String rodeado con las etiquetas <I> </I>, cursivas, del lenguaje HTML. Por ejemplo:

```
var mitexto = "Texto en cursiva";
mitexto = mitexto.italics();
```

Tras la última sentencia la variable mi texto contendrá el valor:

```
<I>Texto en cursiva</I>
```

n) Métodos de String: lastIndexOf(atrcad, desde)

Este método devuelve la primera posición dentro del objeto String donde comienza la subcadena pasada como argumento en **atrcad**, pero realizando la búsqueda de derecha a izquierda. Admite un segundo argumento opcional que indica desde que posición debe realizar la búsqueda, si se omite comienza a buscar por el primer carácter de la derecha, valores negativos o mayores que la longitud de la cadena se consideran 0. Si la subcadena no se encuentra el valor devuelto es -1. Por ejemplo:

```
var cadena = "mi.correo@mail.com";
var arroba = cadena.lastIndexOf('@');
var punto = cadena.lastIndexOf('.',arroba);
```

Este ejemplo devuelve en **arroba** la posición 9 mientras que **punto** contiene la 2 pues la búsqueda se hizo desde la posición donde está el carácter arroba hacia el principio de la cadena encontrando el primer punto. Recuerda que las posiciones en las cadenas se cuentan desde 0.

o) Métodos de String: link(atrcad)

Este método crea, a partir de un objeto String, una cadena conteniendo un elemento ANCHOR del lenguaje HTML, con el atributo HREF igual a la cadena que se le pase en **atrcad**.

```
var enlace = "Dirección" ;
enlace = enlace.link("http://www.ciudadfutura.com');
```

El valor de la variable **enlace** será:

```
<A HREF="http://www.ciudadfutura.com">Dirección</a>
```

La sintaxis de este método permite usar una constante String en lugar del nombre de un objeto String. El ejemplo anterior podría haber escrito como:

```
var enlace = "Dirección".anchor("Refer1");
```

p) Métodos de String: match(expreg)

Este es uno de los más potentes métodos para buscar subcadenas y realizar sustituciones dentro de cadenas de texto. Permite usar patrones de búsqueda construidos con comodines y texto, lo que se denominan expresiones regulares. Este método usa como argumento una expresión regular y va buscando en el objeto alguna subcadena que concuerde con esa expresión. Esta subcadena la devuelve en un **array**. Si no encuentra ninguna devuelve **null**. Además actualiza el valor de una variable global **RegExp** que almacena en sus propiedades diversa información acerca de la búsqueda realizada. Por ejemplo:

```
var frase = new String();
frase="Busco palabras con menos de cinco letras";
var result=new Array();
result=frase.match(/\b\w{1,4}\b/g);
document.write("Hallados: '+result+'<br>');
document.write("En la frase: " + RegExp.input);
```

Si pruebas el ejemplo obtendrás el siguiente listado

```
Hallados: con,de
En la frase: Busco palabras con menos de cinco letras
```

El patrón de búsqueda está encerrado entre dos barras / , y busca caracteres alfanuméricos (\ **w**) comprendidos entre límites de palabras (\ **b**) además hace una búsqueda global (indicado por la **g** fuera de las barras).

q) Métodos de String: replace (expreg, nueva)

A vueltas con las expresiones regulares, difíciles pero potentes. Con este método todas las cadenas que concuerden con la **expreg** del primer argumento son reemplazadas por la cadena especificada en el segundo argumento, **nueva**, que puede contener elementos del patrón mediante los símbolos $1 a $9. El resultado devuelto es la cadena con las sustituciones realizadas. Por ejemplo vamos a cambiar **palabra** por **frase** en la frase "Cada palabra dicha es una palabra falsa"

```
var linea = new String();
linea="Cada palabra dicha es una palabra falsa";
linea = linea.replace(/palabra/g, "frase");
document.write(linea);
```

Si pruebas el ejemplo obtendrás lo siguiente

```
Cada frase dicha es una frase falsa
```

En esta ocasión se ha usado un patrón con el modificador **g** de global por lo que cambia todas las coincidencias, si se omite sólo se cambia la primera. En la cadena nueva pueden usarse elementos del patrón, por ejemplo cambiemos las negritas a cursivas en la frase:

```
var patron = /(<b>)([^<]+)(<\/b>)/g;
var frase = "Cada <b>negrita</b> pasa a <b>itálica</b>";
document.write(frase+"<br>");
newstr = str.replace(patron, "<i>$2</i>");
document.write(frase);
```

veras la frase antes y después del cambio:

```
Cada negrita pasa a itálica
Cada negrita pasa a itálica
```

El $2 es un índice referido a los paréntesis del patrón, así $1 indica lo contenido en el primer paréntesis (<b>) mientras que $3 es <\b>, el tercer paréntesis.

r) Métodos de String: search (expreg)

Es un método similar al método match pero más rápido. Este método realiza una búsqueda en la cadena y devuelve el índice donde se produce la primera concordancia con el patrón o -1 si no existe ninguna. Por ejemplo buscamos las cadenas 'lunes' o 'martes' en la frase cita, la letra i del patrón indica que se debe ignorar el tipo mayúsculas o minúsculas en la búsqueda:

```
var patron = /sábado|miércoles/i;
var cita = "La reunión será el lunes y el miércoles";
document.write(cita.search(patron)+"<br>");
```

Si pruebas el ejemplo obtendrás un 30, la posición de la palabra 'lunes'.

s) Métodos de String: slice (inicio, ultimo)

Este método devuelve la porción de cadena comprendida entre las posiciones dadas por los argumentos inicio y ultimo, o el final de la cadena si se omite este segundo argumento. Si ultimo es negativo, se interpreta como número de posiciones contadas desde el final de la cadena. Si los argumentos no son números enteros, por ejemplo cadenas, se convierten a números enteros como haría el método **Number.parseInt().**

```
var frase = "Autor: Luis Sepúlveda";
var nombre = frase.slice(7);
```

La variable **nombre** contendrá "Luis Sepúlveda". En este otro ejemplo usamos un segundo argumento:

```
var frase = "Autor: Luis Sepúlveda";
var nombre = frase.slice(7, -10);
```

nombre contendrá "Gonzalo"', es decir desde la posición 7 hasta 10 posiciones antes del final.

t) Métodos de String: small()

Este método devuelve una cadena consistente en el objeto String rodeado con las etiquetas <SMALL> </SMALL>, reducir tamaño, del lenguaje HTML. Por ejemplo:

```
var mitexto = "Este es el texto";
mitexto = mitexto.small();
```

Tras la última sentencia la variable mitext contendrá

```
<SMALL>Este es el texto</SMALL>
```

Se puede usar una constante de cadena en lugar de un nombre de variable, así el ejemplo podría haberse escrito:

```
var mitexto = "Este es el texto".small();
```

u) Métodos de String: split (separ)

Devuelve un array conteniendo las porciones en que queda separada la cadena por el separador indicado mediante el argumento **separ**, que será una expresión regular o una cadena literal. Si este separador es una cadena vacía el texto queda desglosado en todos sus caracteres. Si se omite el separador el resultado es un array de un elemento con la cadena completa.

```
var linea=new String("Título: El portero");
var lista = linea.split(/:\s*/);
```

La variable lista es un **array** con dos elementos "Título" y "El portero". También podriamos haberlo escrito como

```
var linea=new String("Título: El portero");
lista = linea.split(":");
document.write(lista);
```

en este caso el primer elemento de lista es "Título" y el segundo " El portero" con un espacio por delante. Por último si el separador es una cadena vacía:

```
var linea=new String("Título: El portero");
lista = linea.split("");
document.write(lista);
```

la variable lista contendrá T,í,t,u,l,o,:, ,E,l, ,p,o,r,t,e,r,o.

v) Métodos de String: strike()

Este método devuelve una cadena consistente en el String rodeado con las etiquetas <STRIKE> </STRIKE>, tachado, del lenguaje HTML. Por ejemplo:

```
var mitexto = "Texto para ver tachado";
mitexto = mitexto.strike();
```

Tras la última sentencia la variable mi texto contendrá el valor:

```
<STRIKE>Texto para ver tachado</STRIKE>
```

w) Métodos de String: sub()

Este método devuelve una cadena consistente en el objeto String rodeado con las etiquetas , subíndice, del lenguaje HTML. Por ejemplo:

```
var mitexto = "Este es el texto";
mitexto = mitexto.sub();
```

Tras la última sentencia la variable mitext contendrá

```
<SUB>Este es el texto</SUB>
```

Se puede usar una constante de cadena en lugar de un nombre de variable, así el ejemplo podría haberse escrito:

```
var mitexto = "Este es el texto".sub();
```

x) Métodos de String: substr(inicio, largo)

Devuelve una subcadena extraída del objeto string comenzando por la posición dada por el primer argumento, **inicio**, y con un número de caracteres dado por el segundo argumento, **largo**. Si se omite este último argumento la subcadena extraída va desde **inicio** hasta el final de la cadena.

```
var linea=new String("Mi página es ideal);
var lista = linea.substr(3);
```

La variable **lista** contendrá "página es ideal".

```
var linea=new String("Mi página es ideal);
var lista = linea.substr(3, 6);
```

ahora la variable **lista** contendrá "página".

y) Métodos de String: substring(ind1,ind2)

Devuelve una subcadena del objeto string que comienza en la posición dada por el menor de los argumentos y finaliza en la posición dada por el otro argumento. Si se omite este último argumento la subcadena extraída va desde la posición indicada por el único argumento hasta el final de la cadena. Si los argumentos son literales se convierten a enteros como un **parseInt()**.

```
var linea=new String("Mi página es ideal);
var lista = linea.substr(3);
```

La variable lista contendrá "página es ideal".

```
var linea=new String("Mi página es ideal);
var lista = linea.substr(3, 9);
```

ahora la variable lista contendrá "página", al igual que en

```
var linea=new String("Mi página es ideal);
var lista = linea.substr(9, 3);
```

z) Métodos de String: sup()

Este método devuelve una cadena consistente en el objeto String rodeado con las etiquetas , superíndice, del lenguaje HTML. Por ejemplo:

```
var mitexto = "Este es el texto";
mitexto = mitexto.sup();
```

Tras la última sentencia la variable **mitexto** contendrá

```
<big>Este es el texto</big>
```

Se puede usar una constante de cadena en lugar de un nombre de variable, así el ejemplo podría haberse escrito:

```
var mitexto = "Este es el texto".sup();
```

aa) Métodos de String: toLowerCase()

Devuelve una cadena igual a la original pero con todos los caracteres en minúsculas. No afecta como es lógico a caracteres no alfabéticos, o sea, a los números, letras acentuadas y caracteres especiales como la Ñ

```
var linea=new String("´Hoy es Domingo");
linea = linea.toLowerCasesubstr();
```

La variable lista contendrá "hoy es domingo".

bb) Métodos de String: toUpperCase()

Devuelve una cadena igual a la original pero con todos los caracteres en mayúsculas. No afecta como es lógico a caracteres no alfabéticos, o sea, a los números, letras acentuadas y caracteres especiales como la Ñ. Es muy útil a la hora de comparar cadenas para asegurarse que dos cadenas no difieran sólo por que algún carácter esté en mayúscula o minúscula.

```
var linea=new String("´Hoy es Domingo");
linea = linea.toUpperCase();
```

La variable lista contendrá "HOY ES DOMINGO".

H) Objeto Date

El objeto Date contiene un valor que representa fecha y hora de un instante dado. Para crear una instancia de este objeto usamos alguna de las siguientes sintaxis:
var fecha= new Date()

```
var fecha= new date(número)
var fecha= new date(cadena)
var fecha=
    new date(año, mes, día[, hora[, minutos[, seg[,ms]]]])
```

Los argumentos encerrados entre corchetes son opcionales. En la primera forma la variable **fecha** contendrá la fecha del día actual. La segunda opción almacena en fecha la fecha dada por el argumento como el número de milisegundos transcurridos desde la media noche del 1 de Enero de 1970. El tercer tipo se usa cuando la fecha se pasa en forma de cadena. Por último la fecha puede crearse pasándole como argumento los números de año, mes, día, hora y opcionalmente, hora, minuto, segundo y milisegundo. Los años posteriores a 1970 puede escribirse con dos dígitos, pero es aconsejable usar siempre cuatro dígitos por aquello de los efectos 2000.

```
var hoy = new date()   /*fecha del día en hoy */
var evento = new Date("November 10 1990");
var otro = new Date("10 Nov 1990");
var otro = new Date("10/02/2000"); //Oct, 2, 2000
var instante = new Date(1990, 11, 10, 20,00);
```

Estas son tres posibles formas de declarar objetos de tipo fecha. Las dos últimas almacenan el mismo día, pero en la última además se guarda la hora.
Donde se usen cadenas para indicar una fecha podemos añadir al final las siglas GMT (o UTC) para indicar que la hora se refiere a hora del meridiano Greenwich, si no se toma como hora local, o sea, según la zona horaria configurada en el ordenador donde se ejecute el script.

Métodos

getDate	**parse**
getDay	**setDate**
getFullYear	**setFullYear**
getHours	**SetHours**
getMilliseconds	**setMilliseconds**
getMinutes	**setMinutes**
getMonth	**setMonth**
getSeconds	**setSeconds**
getTime	**setTime**
getTimezoneOffset	**setYear**
getYear	**toGMT**
Object.toString	**toLocaleString**
Object.valueOf	**toUTCString**

a) Métodos de Date: getDate()

Nos devuelve el día del mes del objeto fecha al que se aplica. Este método controla por supuesto el número de días de cada mes y contempla el caso de años bisiestos, incluida la excepción del 2000. En el siguiente ejemplo se presenta en pantalla Hoy es día 2, suponiendo que la fecha del sistema es 2-10-200. Primero creamos la variable fecha instanciada como un objeto **Date()** para a continuación escribir directamente el valor de **getDate()** aplicado a **fecha**

```
var fecha = new Date();
document.write("Hoy es día: "+fecha.getDate());
```

b) Métodos de Date: getDay()

Nos devuelve el día de la semana del objeto fecha al que se aplica en forma numérica con una cifra entre 0 para el domingo y 6 para el sábado. En el siguiente ejemplo se presenta en pantalla *Hoy es 1*, suponiendo que la fecha del sistema es 2-Octubre-2000, o sea, lunes. Primero creamos la variable fecha instanciada como un objeto **Date()** para a continuación escribir directamente el valor de **getDay()** aplicado a **fecha**

```
var fecha = new Date();
document.write("Hoy es "+fecha.getDay());
```

Si echamos manos de un array podemos mejorar un poquito este ejemplo presentando el nombre del DIA de la semana:

```
var fecha = new Date();
var diaSemana = new Array('domingo', 'lunes', 'martes',
'miércoles', 'jueves', 'viernes','sábado');
var dia = fecha.getDay();
document.write("Hoy es "+diaSemana[dia]);
```

Ahora se obtendría la más amigable frase *Hoy es lunes.*

c) Métodos de Date: getFullYear()

Nos devuelve el año correspondiente del objeto fecha en formato completo es decir incluyendo el siglo. Así si la fecha contiene 2-Octubre-2000, esta función nos dará 2000. Por ejemplo creemos la variable fecha instanciada como un objeto **Date()** para a continuación se presenta directamente el valor de **getFullYear()** aplicado a **fecha**, o sea, 2000.

```
var fecha = new Date();
document.write("El año actual es "+fecha.getFullYear());
```

Este método evita el efecto 2000 al presentar loa años siempre con cuatro dígitos.

d) Métodos de Date: getHours()

Nos devuelve la sección horas en formato 0-24 almacenada en la parte dedicada a la hora del objeto fecha al que se aplica. Así si la fecha contiene 12:40:00, esta función nos dará 12, pero si contiene 5:40:00 nos dará 17. Igualmente el método interpreta los modificadores am / pm pero siempre devuelve la hora en formato de 24 horas. Por ejemplo creemos la variable fecha instanciada como un objeto **Date()**, si son las 17:30:10 el valor de **getHoursr()** presentará 17.

```
var fecha = new Date();
document.write("Son las "+fecha.getHours()+" horas.");
```

Puedes probar que ocurre con otros valores sin necesidad de cambiar la fecha y hora del sistema de la siguiente manera:

```
var fecha = new Date("10-02-2000 08:20:00 pm");
document.write("Son las "+fecha.getHours()+" horas.");
```

Este caso presentará en pantalla *Son las 20 horas*

e) Métodos de Date: getMilliseconds()

Nos devuelve los minutos de la sección dedicada a la hora almacenada en el objeto fecha al que se aplica. Así si la fecha contiene en su parte de hora 12:40:08:640, esta función nos dará 640. Por ejemplo creemos la variable fecha instanciada como un objeto **Date()**, si son las 17:30:08:550 el valor de **getMilliseconds()** presentará 550.

```
var fecha = new Date();
document.write("Son las "+fecha.getHours() );
document.write(":" + fecha.getMinutes() );
document.write(":" + fecha.getSeconds() );
document.write(":" + fecha.getMilliseconds());
```

Esta función está presente en JScript de Microsoft y en ECMAScript pero no es soportada por Netscape.

f) Métodos de Date: getMinutes()

Nos devuelve los minutos de la sección dedicada a la hora almacenada en el objeto fecha al que se aplica. Así si la fecha contiene en su parte de hora 12:40:08, esta función nos dará 24. Por ejemplo creemos la variable fecha instanciada como un objeto **Date()**, si son las 17:30:08 el valor de **getMinutes()** presentará 8.

```
var fecha = new Date();
document.write("Son las "+fecha.getHours() );
document.write(":" + fecha.getMinutes() );
document.write(":" + fecha.getSeconds() ); ;
```

Si queremos que quede más presentable podemos completar con ceros por la izquierda cuando el número (de horas, minutos o segundos) sea menor que 10. Esto es tan fácil como se ve en el ejemplo:

```
var fecha = new Date();
var horas = fecha.getHours();
var mins = fecha.getMinutes();
var segs = fecha.getSeconds();
horas = (horas < 10)?"0"+horas:horas;
mins = (mins < 10)?"0"+mins:mins;
segs = (segs<10)?"0"+segs:segs;
document.write("Son las "+horas);
document.write(":" + mins);
document.write(":" + segs);
```

g) Métodos de Date: getMonth()

Nos devuelve en forma numérica el mes correspondiente al objeto fecha al que se aplica. Así para la fecha correspondiente al 10/Oct/2000, esta función nos dará 10, el número de orden de Octubre. Por ejemplo creemos la variable fecha instanciada como un objeto **Date()**,

```
var fecha = new Date();
document.write("Este mes es el "+fecha.getMonth() );
```

Si queremos que aparezca el nombre del mes en lugar del número debemos crear primero un **array** de doce elementos y rellenarlos con los nombres de los meses, luego usamos el resultado de **getMonth()** como índice a ese array

```
var array = new meses();
var fecha = new Date();
var nmes = fecha.getMonth();
mes[1] = "Enero";
mes[2] = "Febrero";
mes[3] = "Marzo";
mes[4] = "Abril";
... ...
document.write("Mes actual:" + meses[nmes]);
```

h) Métodos de Date: getSeconds()

Nos devuelve los segundos de la sección dedicada a la hora almacenada en el objeto fecha al que se aplica. Así si la fecha contiene en su parte de hora 12:40:08, esta función nos dará 8. Por ejemplo creemos la variable fecha instanciada como un objeto **Date()**, si son las 17:30:08 el valor de **getSeconds()** presentará 8.

var fecha = new Date();

```
document.write("Son las "+fecha.getHours() );
document.write(":" + fecha.getMinutes() );
document.write(":" + fecha.getSeconds() ); ;
```

Si queremos que quede mas presentable podemos completar con ceros por la izquierda cuando el número (de horas, minutos o segundos) sea menor que 10. Esto es tan fácil como se ve en el ejemplo:

```
var fecha = new Date();
var horas = fecha.getHours();
var mins = fecha.getMinutes();
var segs = fecha.getSeconds();
horas = (horas < 10)?"0"+horas:horas;
mins = (mins < 10)?"0"+mins:mins;
segs = (segs<10)?"0"+segs:segs;
document.write("Son las "+horas);
document.write(":" + mins);
document.write(":" + segs);
```

i) Métodos de Date: getTime()

Nos devuelve la cantidad de milisegundos transcurridos desde el 1 de Enero de 1970 hasta la hora almacenada en el objeto fecha al que se aplica. En el ejemplo que sigue creamos un objeto **Date** con la fecha actual, a continuación escribimos el número de milisegundos dado por este función, verás que este número habitualmente es muy grande, realmente esta función puede ser útil para calcular el tiempo transcurrido entre dos instantes, por ejemplo en un puzzle podría ser útil para calcular el tiempo que el jugador emplea en resolver el puzzle, restando el **getTime()** obtenido al final del juego del **getTime()** obtenido al inicio.

```
var ahora = new Date();
document.write(ahora.getTime());
```

j) Métodos de Date: getTimezoneOffset()

Esta función nos da la diferencia horaria en minutos del ordenador con respecto al meridiano de Greenwich. El valor depende por tanto de la zona o huso horario para el que esté configurado el ordenador, pudiendo ser negativo o positivo según esté en la zona oriental u occidental. El ejemplo que muestra el uso de la función define la variable **ahora** con la fecha actual y devuelve en minutos la diferencia horaria con la GMT, el resultado depende de tu ordenador.

```
var ahora = new Date();
document.write(ahora.getTimezoneOffset());
```

k) Métodos de Date: getYear()

Nos devuelve en forma numérica el mes correspondiente al objeto fecha al que se aplica. Así para la fecha correspondiente al 10/Oct/2000, esta función nos dará 2000 en IExplorer y 100 en Netscape. Por ejemplo creamos la variable fecha instanciada como un objeto **Date()**, y luego extraemos el año

```
var fecha = new Date();
document.write("Este año es el "+fecha.getYear());
```

Si pruebas este ejemplo en Netscape y en Internet Explorer verás que éste último da el año con cuatro dígitos mientras que el primero elimina el siglo.

l) Métodos de Date: Object.toString()

m) Métodos de Date: Object.valueOf()

Los casos Object.toString, y Object.valueOf, ya fueron explicados en el apartado **E) Objeto Object.**

n) Métodos de Date: parse(fecha)

Nos devuelve la cantidad de milisegundos transcurridos desde el 1 de Enero de 1970 00:00:00 hasta la hora pasada en el argumento **fecha** como una cadena de caracteres. Este método es un método global del objeto y por tanto no es necesario crear un objeto **Date** para usarlo, como vemos en este ejemplo.

```
var transcurridos = Date.parse("1/1/2000 00:00:00");
document.write(transcurridos);
```

o) Métodos de Date: setDate(diames)

Nos permite cambiar el día del mes del objeto fecha al que se aplica para poner el valor que se pasado en el argumento **diames**. Este método controla por supuesto el número de días de cada mes y contempla el caso de años bisiestos, incluida la excepción del 2000, de forma que si pasamos como argumento 31 y el mes es de 30 días la función corrige la fecha completa pasándola al día 1 del mes siguiente. Esto lo vemos en el ejemplo que sigue: creamos una variable como un objeto **Date** con el último día de Septiembre (mes de 30 días) e intentamos poner el día a 31, luego comprobamos la fecha almacenada:

```
var fecha = new Date("1 Sep 2000");
fecha.setDate(31);
document.write("Hoy es día: "+fecha.toString());
```

Como verás si pruebas el ejemplo la fecha es corregida y pasa a 1 de Octubre.

p) Métodos de Date: setFullYear()

Nos permite cambiar el año del objeto fecha por el valor pasado como argumento, un número interpretado como año completo, o sea, que para poner el año 1995 se debe pasar 1995, no el 95. El ejemplo pone precisamente este valor en el campo año de la variable **fecha**.

```
var fecha = new Date();
fecha.setFullYear(1995)
document.write(fecha.toString());
```

Como el año es de cuatro dígitos no hay problema de efecto 2000.

q) Métodos de Date: setHours()

Nos permite modificar la hora almacenada en el objeto fecha al que se aplica y poner la que se pasa como argumento. Lógicamente este argumento estará entre 0 y 24, aunque si se usa un valor fuera de este rango la fecha es corregida en consecuencia. Por ejemplo si intentamos poner la hora en 30 la fecha se adelanta 24 horas y se pone en las 6 horas, cambiando además el día. Igualmente si se usa un número negativo en el argumento se toma como horas antes de la última media noche del mismo día. Observa todo esto en el ejemplo, donde al final de cada acción se presenta la fecha completa en forma de cadena:

```
var fecha = new Date("10 Sep 2000 00:00:00");
var nl="<br>";
fecha.setHours(20);
document.write("A las 20: "+fecha.toString()+nl);
fecha.setHours(30);
document.write("A las 30: "+fecha.toString()+nl);
fecha.setHours(-2);
document.write("A las -2: "+fecha.toString()+nl);
```

r) Métodos de Date: setMilliseconds()

Nos permite modificar el número de milisegundos de la hora almacenada en el objeto fecha al que se aplica, poniendo los milisegundos al valor pasado como argumento. Habitualmente el argumento estará comprendido entre 0 y 1000.

```
var fecha = new Date("10 Sep 2000 00:00:00");
var nl="<br>";
fecha.setMilliSeconds(900);
document.write(fecha.toString()+nl);
```

s) Métodos de Date: setMinutes(minact)

Nos permite ajustar los minutos de la sección dedicada a la hora almacenada en el objeto fecha al que se aplica. El nuevo valor para los minutos se pasa como argumento, que habitualmente estará entre 0 y 59, aunque un valor fuera de este rango no da error sino que ajusta el resto de la hora. Así 68 en el argumento adelanta el reloj una hora pone los minutos a 8, mientras que un -4 pone los minutos a 56 (60 menos 4). Puedes ver lo que ocurre en este ejemplo

```
var fecha = new Date("10 Sep 2000 00:00:00");
var nl="<br>";
fecha.setMinutes(20);
document.write("Minact 20: "+fecha.toString()+nl);
```

```
fecha.setMinutes(68);
document.write("Minact 68: "+fecha.toString()+nl);
fecha.setMinutes(-4);
document.write("Minact -4: "+fecha.toString()+nl);
```

Como ves si es necesario, se ajusta la hora cuando el número de minutos supera el valor 59

t) Métodos de Date: setMonth(nummes)

Esta función se usa para modificar el mes del objeto fecha al que se aplica. El nuevo valor se pasa como un número en el argumento. El valor deberá ser como es lógico numérico o convertible a numérico y comprendido entre 0 (Enero) y 11 (Diciembre). Si el valor está fuera del rango se toma el exceso sobre 11 y se corrige adecuadamente la fecha, y si es negativo se toma como número de meses antes de Enero (-1 sería Diciembre, -2 sería Noviembre, etc.). El ejemplo es muy sencillo, en este caso se cambia el mes de Septiembre por Marzo

```
var fecha = new Date("10 Sep 2000 00:00:00");
fecha.setMonth(2);
document.write("Minact 20: "+fecha.toString());
```

u) Métodos de Date: setSeconds(miliseg)

Nos permite modificar el número de segundos de la hora almacenada en el objeto fecha al que se aplica, poniendo los segundos al valor pasado como argumento. Habitualmente el argumento estará comprendido entre 0 y 60.

```
var fecha = new Date("10 Sep 2000 00:00:00");
var nl="<br>";
fecha.setSeconds(90);
document.write(fecha.toString()+nl);
```

v) Métodos de Date: setTime()

Nos devuelve la cantidad de milisegundos transcurridos desde el 1 de Enero de 1970 hasta la hora almacenada en el objeto fecha al que se aplica. En el ejemplo que sigue creamos un objeto **Date** con la fecha actual, a continuación escribimos el número de milisegundos dado por este función, verás que este número habitualmente es muy grande, realmente esta función puede ser útil para calcular el tiempo transcurrido entre dos instantes, por ejemplo en un puzzle podría ser útil para calcular el tiempo que el jugador emplea en resolver el puzzle, restando el **setTime()** obtenido al final del juego del **setTime()** obtenido al inicio.

```
var ahora = new Date();
document.write(ahora.setTime());
```

x) Métodos de Date: setYear()

Nos permite cambiar el año del objeto fecha por el valor pasado como argumento, un número interpretado como año completo, o sea, que para poner el año 1995 se debe pasar 1995, no el 95. El ejemplo pone precisamente este valor en el campo año de la variable **fecha**.

```
var fecha = new Date();
fecha.setFullYear(1995)
document.write(fecha.toString());
```

Como el año es de cuatro dígitos no hay problema de efecto 2000.

y) Métodos de Date: toLocaleString()

Esta función se usa para transformar el objeto fecha al que se aplica a una cadena de caracteres según el estándard UTC (Universal Time Coordinates), denominación actual del GMT (Greenwich Meridian Time). La hora se ajusta según la configuración del ordenador. En el ejemplo que sigue la cadena devuelta será "Mon, 10 Apr 2000 02:00:00 UTC" (Netscape cambia UTC por GMT)

```
var fecha = new Date("10 Apr 2000 02:00:00");
document.write(fecha.toUTCString());
```

Como ves existe una diferencia en la hora almacenada y la devuelta por la función, esto es debido a que la cadena devuelta es la hora correspondiente a Greenwich, no la local del ordenador.
Existe una función similar, la **toGMTString()**, que es considerada como obsoleta y que se mantiene por cuestiones de compatibilidad.

z) Métodos de Date: toUTCString(fecha)

Nos devuelve la cantidad de milisegundos transcurridos desde el 1 de Enero de 1970 00:00:00 hasta la hora pasada en el argumento **fecha**. Este argumento se pasa como una serie de números separados por comas en el orden: Año, mes, día, y opcionalmente: hora, minuto, segundos. Este método es un método global del objeto y por tanto no es necesario crear un objeto **Date** para usarlo, como vemos en este ejemplo que toma como fecha actual el 1 de Noviembre de 2000 a las 00:00:00.

```
var transc= Date.UTC(2000,10,1);
document.write(transc);
```

I) Objeto Math

Es el objeto que usa JavaScript para dotar al lenguaje de funciones matemáticas avanzadas y de constantes predefinidas, como el número PI.

Métodos

abs: Valor absoluto	**cos**: coseno	**pow**: Potencia de
acos: Arco coseno	**exp**: Exponencial	**random**: Número al azar
asin: Arco seno	**floor**: Redondeo inferior	**round**: Redondear
atan: Arco tangente	**log**: Logaritmo natural	**sin**: Seno
atan2: Arco tangente	**max**: máximo	**sqrt**: Raíz cuadrada
ceil: Redondeo superior	**min**: Mínimo	**Tan**: Tangente

Propiedades
Son las habituales constantes como el número e, PI y algunos otros valores habituales en cálculos matemáticos.

E Constante de Euler la base para los logaritmos naturales	**LN10** Logaritmo natural de 10	**LOG10E** Logaritmo en base 10 de E	**SQRT1_2** Raíz cuadrada de 0.5 o sea la inversa dela raíz de 2
LN2 Logaritmo natural de 2	**LOG2E** Logaritmo en base 2 de E	**PI** El conocido número pi	**SQRT2** Raíz cuadrada de 2

a) Métodos Math: abs(exprnum)

Devuelve el valor absoluto, o sea, sin signo, del argumento. Si el argumento fuera no entero será convertido a numérico siguiendo las reglas de la función **parseInt()** o **parseFloat()**. Su sintaxis es tan simple como el ejemplo:

```
var numabs = Math.abs( - 45)
```

la variable **numabs** contendrá el valor 45.

b) Métodos Math: acos(exprnum)

Es una función trigonométrica que sirve para calcular el ángulo cuyo coseno es el valor dado por el argumento, el arccos(). Este argumento deberá ser una expresión numérica o transformable en numérica, comprendida entre -1 y +1 y el ángulo devuelto viene dado en radianes.

```
var arco = Math. acos( 1)
```

la variable **arco** contendrá el valor 0.

Recuerda las matemáticas del cole. El radián es una unidad de medida de arcos tal que 2*pi radianes equivalen a 360º.

c) Métodos Math: asin(exprnum)

Es una función trigonométrica que sirve para calcular el ángulo cuyo seno es el valor dado por el argumento, es decir, el llamado arcosen. Este argumento deberá ser una expresión numérica, o transformable en numérica, comprendida entre -1 y +1 y el ángulo devuelto viene dado en radianes.

```
var arco = Math.asin( 1 )
```

la variable **arco** contendrá el arco cuyo seno es 1, o sea, 1.57 o lo que es lo mismo pi / 2 radianes.

Recuerda las matemáticas del cole. El radián es una unidad de medida de arcos tal que 2*pi radianes equivalen a 360º.

d) Métodos Math: atan(exprnum)

Es una función trigonométrica que sirve para calcular el ángulo cuya tangente es el valor dado por el argumento, o sea el arctg(). Este argumento deberá ser una expresión numérica o transformable en numérica, sin límites, y el ángulo devuelto viene dado en

radianes.

```
var arco = Math.atan( 1 )
```

la variable **arco** contendrá el arco cuya tangente es 1, o sea, 0.7853981633974483 o lo que es lo mismo pi / 4 radianes (45º).

Recuerda las matemáticas del cole. El radián es una unidad de medida de arcos tal que 2*pi radianes equivalen a 360º.

e) Métodos Math: atan2(coorX, coorY)

Es una función trigonométrica que sirve para calcular el ángulo cuya tangente es el cociente de sus argumentos, en otras palabras devuelve el ángulo desde el origen de coordenadas hasta un punto cuyas coordenadas son los argumentos de la función. Los argumentos deberán ser numéricos o transformables en numéricos, y el ángulo devuelto viene dado en radianes.

```
var argum= Math.atan2( 10, 4)
```

la variable **argum** contendrá el arco cuya tangente es 10/4.

Recuerda las matemáticas del cole. El radián es una unidad de medida de arcos tal que 2*pi radianes equivalen a 360º . Es una función útil para trabajar con números complejos pues realmente calcula el argumento de un complejo donde **coorY** es la parte real y **coorX** es la imaginaria.

f) Métodos Math: ceil(exprnum)

Devuelve el valor del argumento redondeado por exceso, es decir el menor número entero mayor o igual al argumento. Si el argumento fuera no numérico será convertido a numérico siguiendo las reglas de la función **parseInt()** o **parseFloat()**. Su sintaxis es tan simple como el ejemplo:

```
var redexceso = Math.ceil( 4.25)
```

la variable **redexceso** contendrá el valor 5.

g) Métodos Math: cos(exprnum)

Es una función trigonométrica que sirve para calcular el coseno del ángulo pasado como argumento en radianes. Este argumento deberá ser una expresión numérica o transformable en numérica.

```
var coseno = Math.cos( Math.PI/2)
```

la variable **coseno** contendrá el valor 0, que es el coseno de pi/2 radianes (90º).

h) Métodos Math: exp(exprnum)

Devuelve el valor del número e (constante de Euler, aproximadamente 2,178) elevada al exponente dado por el argumento. Si el argumento fuera no entero será convertido a numérico siguiendo las reglas de las funciones **parseInt()** o **parseFloat()**. Su sintaxis es tan simple como el ejemplo:

```
var e4 = Math.exp(4)
```

la variable **e4** contendrá el valor e^4. El número e es la base de los logaritmos neperianos por lo que esta función sirve para calcular antilogaritmos.

i) Métodos Math: floor(exprnum)

Devuelve el valor del argumento redondeado por defecto, es decir, el mayor número entero menor o igual al argumento. Si el argumento fuera no numérico será convertido a numérico siguiendo las reglas de la función **parseInt()** o **parseFloat()**. Su sintaxis es tan simple como el ejemplo:

```
var redexceso = Math.floor( 4.75)
```

la variable **redexceso** contendrá el valor 4.

j) Métodos Math: log(exprnum)

Devuelve el logaritmo natural o neperiano, o sea, en base al número **e**, del argumento. Si el argumento fuera no numérico será convertido a numérico siguiendo las reglas de la función **parseInt()** o **parseFloat()**. Si el argumento fuera un valor negativo esta función devuelve **NaN**. Su sintaxis es tan simple como el ejemplo:

```
var logaritmo = Math.log( 1000)
```

la variable **logaritmo** contendrá el valor 6.907755278982137 .

k) Métodos Math: max(num1, num2)

Devuelve el mayor de los dos números o expresiones numéricas pasadas como argumentos. Si alguno de los argumentos fuera no numérico será convertido a numérico siguiendo las reglas de la función **parseInt()** o **parseFloat()**. Su sintaxis es tan simple como el ejemplo:

```
var mayor = Math.wax( 12, 5)
```

la variable **mayor** contendrá el valor 12.

l) Métodos Math: min(num1, num2)

Devuelve el menor de los dos números o expresiones numéricas pasadas como argumentos. Si alguno de los argumentos fuera no numérico será convertido a numéricos siguiendo las reglas de la función **parseInt()** o **parseFloat()**. Su sintaxis es tan simple como el ejemplo:

```
var menor = Math.min( 12, 5)
```

la variable **menor** contendrá el valor 5.

m) Métodos Math: pow(base, exp)

Calcula la potencia de un número, dado por el argumento **base**, elevado al exponente dado por el argumento **exp**. Si alguno de los argumentos fuera no numérico será convertido a numérico siguiendo las reglas de la función **parseInt()** o **parseFloat()**. Su sintaxis es tan simple como el ejemplo:

```
var potencia = Math.pow( 2, 4)
```

la variable **potencia** contendrá el valor 16.

n) Métodos Math: random()

Calcula un número aleatorio, realmente seudo-aleatorio, comprendido entre 0 y 1 ambos inclusive. Cada vez que se carga el intérprete de JavaScript se genera una semilla base para el cálculo. No lleva argumentos y su sintaxis es tan simple como el ejemplo:

```
var azar = Math.random()*10
```

la variable **azar** contendrá un número al azar entre 0 y 10.

o) Métodos Math: round(exprnum)

Devuelve el valor entero mas próximo al número pasado como argumento, es decir, redondea. Si la parte decimal del argumento es 0.5 o mayor devuelve el primer entero por encima del argumento (redondeo por exceso) en caso contrario devuelve el entero anterior al argumento (redondeo por defecto). Si el argumento fuera no entero será convertido a numérico siguiendo las reglas de la función **parseInt()** o **parseFloat()**. Su sintaxis es tan simple como el ejemplo:

```
var entero1 = Math.random(4.25)
var entero2 = Math.random(4.65)
```

la variable **entero1** contendrá el valor 4 mientras **entero1** que contendrá 5.

p) Métodos Math: sin(exprnum)

Es una función trigonométrica que sirve para calcular el seno del ángulo pasado como argumento en radianes. Este argumento deberá ser una expresión numérica o transformable en numérica.

```
var seno = Math.sin( Math.PI/2)
```

la variable **seno** contendrá el valor 1, que es el seno de pi/2 radianes (90º).

q) Métodos Math: sqrt(exprnum)

Devuelve la raíz cuadrada del valor pasado como argumento. Si el argumento fuera no entero será convertido a numérico siguiendo las reglas de la función **parseInt()** o **parseFloat()**. Si el argumento fuera negativo o cualquier otro valor no numérico devolverá **NaN**. Su sintaxis es tan simple como el ejemplo:

```
var raiz = Math.sqrt( 49)
```

la variable **raiz** contendrá el valor 7.

r) Métodos Math: tan(exprnum)

Es una función trigonométrica que sirve para calcular la tangente del ángulo pasado como argumento en radianes. Este argumento deberá ser una expresión numérica o transformable en numérica.

```
var tangente = Math.tan( Math.PI/4)
```

la variable **tangente** contendrá el valor 1, que es la tangente de pi/4 radianes (45º).

6) Ejemplos JavaScript:

Estos ejemplos te permitirán poner en práctica los conceptos aprendidos con el tutor. Te bastará con un editor HTML para probar por ti mismo estos programas de muestra. Sólo se usan dos métodos del navegador: **document.write** y ventanas **alert** para mostrar resultados. Recuerda que el código lo debes situar en la sección HEAD de la página y las llamadas a las funciones en el cuerpo. Por ahora no se usan **eventos**, eso queda para la 2ª Fase: el HTML dinámico.

Ejemplos

Operadores	**Comprobar E-mail 2**
Media Aritmética	**Buscar en un Array**
Saludo	**Extraer Subcadena**
Array aleatorio	**Creando Objetos**
Comprobar E-mail 1	

A) Operadores

Enunciado

Se trata de crear una sencilla función para sumar dos números, que se le pasan como argumentos. La función devolverá la suma de ambos números.

El código

Sección **Head**

```
<SCRIPT TYPE="text/javascript" LANGUAGE="JavaScript">
<!--
function sumar(sum1, sum2)
{
    var resultado;
    resultado = sum1 + sum2;
    return resultado;
}
-->
</SCRIPT>
```

Sección **Body**

```
<SCRIPT TYPE="text/javascript" LANGUAGE="JavaScript">
<!--
document.write("4 + 10 = "+ sumar(4, 10) ); /*Mostrar resultado*/
-->
</SCRIPT>
```

Comentarios

En este sencillo ejemplo creamos la función mediante la palabra clave **function** a la que sigue el nombre en este caso **sumar**. Entre paréntesis escribimos los nombres que damos a los argumentos que usa la función: **sum1** y **sum2** en el ejemplo. Cuando llamemos a la función entre paréntesis escribiremos dos valores que se guardarán en estas variables. También vemos como crear una variable usando la palabra clave **var**

seguida del nombre de la variable, en el ejemplo **resultado**. El símbolo = es el operador asignación con el que guardamos la suma en la variable **resultado**. Por último usamos **return** para devolver el **resultado** al programa que usa la función **sumar**. Los símbolos /* y */ encierran cualquier comentario personal para clarificar el programa.
En el cuerpo de la página usaremos **document.write** para mostrar en una ventana emergente el resultado de la suma. Una última observación que será común para todos los ejemplos: observa como todo lo que sea código JavaScript se encierra entre las etiquetas <Script....> y </script>, esto le indica al navegador que debe ejecutar instrucciones de programa. Igualmente verás los símbolos <!-- y --> como sabes es la manera que usa HTML para señalar los comentarios, y en este caso se usa para que los navegadores que no soportan JavaScript se salten el código.

B) Media Aritmética

Enunciado

Crear una función que calcule la media aritmética (suma de datos dividido por número de valores) de un conjunto de datos numéricos que se le pasan como argumentos. El número de datos es variable.

El código

Sección **Head**

```
<SCRIPT TYPE="text/javascript" LANGUAGE="JavaScript">
<!--
function media()
{
  var total=0, res, numDatos, item;
  numDatos = arguments.length;
  for (item = 0; item < numDatos; item++)
    {
     total += arguments[item];
    }
  res = total/numDatos;
  return res;
}
-->
</SCRIPT>
```

Sección **Body**

```
<SCRIPT TYPE="text/javascript" LANGUAGE="JavaScript">
<!--
document.write( media( 3,50,40,25,10) );
-->
</SCRIPT>
```

Comentarios

Se crea la función como es habitual con **function**. Se declaran las variables que se van a usar, en esta ocasión al mismo tiempo inicializamos el valor de una de ellas, **total**, que va a almacenar la suma de los datos.
Guardamos en numDatos el número de argumentos pasados mediante la propiedad **length** de la propiedad **arguments** del objeto **Function**.
Mediante un bucle **for** recorremos la matriz argumentos y vamos acumulando en **total**

los datos pasados a la función. Observa el uso del operador **+=** que suma al contenido de la variable total el valor del elemento **arguments[item]**.
Calcula la media dividiendo **total** por el **numDatos** y por último devuelve el resultado almacenado en **res**.
Este código podría ser mas corto pero he preferido detallarlo lo más posible para ilustrar el uso de operadores y hacer el programa mas claro.

C) Saludo

Enunciado

Este programa hará que tu ordenador te salude educadamente dándote los buenos días, buenas tardes o buenas noches dependiendo de la hora que sea.

El código

Sección **Head**

```
<SCRIPT TYPE="text/javascript" LANGUAGE="JavaScript">
<!--
function saludar(){
var tiempo = new Date();
var hora, cad="son las ";
with (tiempo){
hora = getHours();
cad += hora + ":" + getMinutes()+":"+getSeconds();
}
if (hora < 12)
     cad = "Buenos días, "+ cad;
else if (hora < 18)
     cad = "Buenas tardes, "+ cad;
else
     cad = "Buenas noches, " + cad;

return cad
}
// -->
</SCRIPT>
```

Sección **Body**

```
<SCRIPT TYPE="text/javascript" LANGUAGE="JavaScript">
<!--
document.write(saludar());
-->
</SCRIPT>
```

Comentarios

En este ejemplo usamos el objeto **Date** para determinar la hora del día, a continuación extraemos la hora del objeto tiempo y construimos una cadena **cad** con la hora, minuto y segundo del día para el saludo. La variable **hora** la comparamos con las 12 y las 18 mediante sentencias **if...else** anidadas. Si es antes de las 12 (**hora** < 12) el saludo a escribir será la cadena "Buenos días, son las..." seguido de la hora completa que está almacenada en la variable **cad**, si no es antes de las 12 comprobamos si es antes de las 18 si es así la cadena es "Buenas tardes son las..." y si no se da ninguno de los casos anteriores debe ser por la noche. Observa el uso de la estructura **with** que permite

trabajar con los elementos de un objeto sin tener escribir su nombre y el punto separador (tiempo.getDate() etc). Observa también como modificamos la variable **cad** anteponiéndole al valor que guarda (la hora completa) el prefijo con el saludo adecuado.

D**)** Array aleatorio

Enunciado

Este sencillo ejemplo trata de como rellenar una lista de longitud dada, en este caso 10, con números aleatorios comprendidos entre 0 y 20, todos enteros.

El código

Sección **Head**

```
<SCRIPT TYPE="text/javascript" LANGUAGE="JavaScript">
<!--
function rellenar(matriz)
{
var largo, valor, ind;
largo = matriz.length;
for(ind = 0; ind < largo; ind++)
   {
    valor = Math.random()*20;
    matriz[ind] = Math.round(valor);
   }
}
// -->
</SCRIPT>
```

Sección **Body**

```
<SCRIPT TYPE="text/javascript" LANGUAGE="JavaScript">
<!--
var lista=new Array(10);
rellenar(lista);
document.write(lista);
// -->
</SCRIPT>
```

Comentarios

En este programa es interesante observar como los valores dados al argumento dentro de la función permanecen al finalizar ésta, de forma que la variable **lista** antes de pasar por la función estaba vacía y después de ejecutarse **rellenar** posee valores numéricos aleatorios. Esto ocurre así porque ese argumento es un objeto **Array**, es el único caso en que JavaScript usa paso de argumentos por referencia. Otro punto es que esta función no usa **return** ya que no devuelve ningún valor. El resto del código es muy simple: un bucle **for()** para recorrer la matriz e irle dando valores calculados con el método **random** del objeto **Math** redondeados al entero más próximo mediante el método **round**.

E) Comprobar E-mail 1

Enunciado

Esta es una función que puede resultar bastante útil para comprobar direcciones de correo. Se trata de ver si una dirección de email contiene el carácter @ y el punto en su lugar.

El código

Sección **Head**

```
<SCRIPT TYPE="text/javascript" LANGUAGE="JavaScript">
<!--
function checkemail(email)
{
var ind1, ind2, ind3;
ind1 = email.indexOf('@');
ind2 = email.indexOf('.');
ind3 = email.lastIndexOf('@');
if ((ind1<=0) || (ind2<ind1) || (ind3 != ind1))
      return "No es correcto";
else
     return "Correcto";
}
// -->
</SCRIPT>
```

Sección **Body**

```
<SCRIPT TYPE="text/javascript" LANGUAGE="JavaScript">
<!--
var dire = "javascript@ciudadfutura.com";
document .write(dire+"  "+checkemail(dire));
dire= "jrasr@rest.est@es";
document.write(dire+"  "+checkemail(dire));
-->
</SCRIPT>
```

Comentarios

Sin comentarios que destacar.

F) Comprobar E-mail 2

Enunciado

Esta es una función que puede resultar bastante útil para comprobar direcciones de correo. Se trata de ver si una dirección de email contiene caracteres correctos y sigue el esquema habitual: **usuario@servidor**, usuario y servidor podrán contener caracteres alfanuméricos mas _ (guión bajo) y - (guión normal).

El código

Sección **Head**

```
<SCRIPT TYPE="text/javascript" LANGUAGE="JavaScript">
<!--
function checkMail(cadena) {
var plant = /[^\w^@^\.^-]+/gi
if (plant.test(cadena))
     alert(cadena + " contiene caracteres extraños.")
else{
     plant =/(^\w+)(@{1})([\w\.-]+$)/i
     if (plant.test(cadena))
         alert(cadena + " es correcta.")
     else
         alert(cadena + " no es válida.")
}
}
// -->
</SCRIPT>
```

Sección **Body**

```
<form name="form1" method="post" action="">
Introduce tu e-mail
<input type="text" name="email">
<input type="button" name="Button" value="Comprobar"
onclick="checkMail(this.form.email.value)">
</form>
```

Comentarios

La función checkMail() comprueba primero si existe algún carácter extraño para lo cual usa la expresión regular: /[^\w^@^\.^-]+/gi , comprueba si existe algún carácter no alfanumérico, o diferente de @, del punto o del guión. Una vez ha hecho esto pasa a comprobar la estructura de la cadena mediante otra expresión regular, en este caso comprueba que la cadena comienza con un carácter alfanumérico (^ indica aquí inicio de cadena y \w caracteres alfanuméricos) seguido de otros similares, este grupo debe ir seguido por un sólo símbolo arroba (@{1}) tras el cual puede existir cualquier grupo de alfanuméricos incluido el punto o el guión ([\w\.-]) hasta llegar al final de la cadena ($). Si tienes dudas consulta el apartado descriptivo de las expresiones regulares y el dedicado a sus métodos.

G**)** Buscar en un Array

Enunciado

En esta ocasión tenemos una lista de nombres y deseamos determinar si un nombre en concreto está en la lista, y si es así en que posición se encuentra. Si el nombre no está en la lista la función debe devolver un valor negativo.

El código

Sección **Head**

```
<SCRIPT TYPE="text/javascript" LANGUAGE="JavaScript">
<!--
function buscarItem(lista, valor){
```

```
var ind, pos;
for(ind=0; ind<lista.length; ind++)
   {
    if (lista[ind] == valor)
      break;
    }
pos = (ind < lista.length)? ind : -1;
return (pos);
}
// -->
</SCRIPT>
```

Sección **Body**

```
<SCRIPT TYPE="text/javascript" LANGUAGE="JavaScript">
<!--
var lista1 = new Array('Juan', 'Pedro', 'Luis', 'María', 'Julia');
var cad = 'María';
var pos = buscarItem(lista1, cad);
if (pos >=0)
   document.write(cad+' está en la posición '+ pos );
else
   document.write(cad+ ' no está.');
// -->
</SCRIPT>
```

Comentarios

La lista se construye como un **Array** en el que almacenamos nombres. La variable **cad** contiene el nombre que deseemos buscar. La función funciona con dos argumentos: la **lista** de nombres y el **valor** a buscar, en ella hacemos un bucle para recorrer la variable **lista** hasta encontrar el nombre buscado, si lo encontramos detenemos el bucle con la instrucción **break**. Si el bucle ha terminado sin encontrar el nombre la variable **ind** será igual a la longiutd del Array (recuerda que los índices van desde 0 a la longitud del array menos 1), mientras que si se ha encontrado el nombre su valor será menor que esa longitud.. Observa la sentencia después del bucle: en **pos** ponemos el valor del ind si se ha encontrado el nombre y si no ponemos -1, es la forma abreviada de usar una sentencia **if...else**.

H**)** Extraer subcadena

Enunciado

Partimos de un texto y veremos como leer una parte del mismo. La parte a leer va a estar delimitada por dos etiquetas una de principio y otra de fin. La etiqueta de inicio será /* y la de fin será */.

El código

Sección **Head**

```
<SCRIPT TYPE="text/javascript" LANGUAGE="JavaScript">
<!--
function extraer(texto, etqini, etqfin)
{
var ind0, ind1, parte ="";
ind0 = texto.indexOf(etqini);
if (ind0 >=0)
```

```
{
ind1 = texto.indexOf(etqfin);
if (ind1>ind0)
parte = texto.substring(ind0+etqini.length, ind1);
}
return parte;
}
// -->
</SCRIPT>
```

Sección **Body**

```
<SCRIPT TYPE="text/javascript" LANGUAGE="JavaScript">
<!--
var cadena = "Esta cadena es la que, /*en el ejemplo*/, se va a
procesar";
document.write(cad);
document.write('<br>Cadena extraida:<br>');
document.write(extraer(cadena, '/*','*/'));
// -->
</SCRIPT>
```

Comentarios

En este ejemplo vemos como extraer de un texto una parte delimitada por etiquetas. En primer lugar buscamos la posición de la etiqueta de inicio, **etqini**, que guardamos en **ind0**; si existe esta etiqueta (ind0 >= 0) buscamos la etiqueta de final, **etqfin**, que debe estar después de la de inicio, por eso comparamos (ind1> ind0). Por último extraemos los caracteres comprendidos entre el final de la **etqini** (**ind0**+ longitud de **etqini**) y el inicio de la **etqfin** usando el método **substring**.

I) Creando Objetos

Enunciado

Vamos a crear un objeto usado para representar un artículo de una tienda. El artículo se va a caracterizar por una descripción, un código y un precio, y debe permitir el cálculo de su correspondiente IVA.

El código

Sección **Head**

```
<SCRIPT TYPE="text/javascript" LANGUAGE="JavaScript">
<!--
function iva()
{
return Math.round(this.valor*0.16);
}
function total(reb)
{
var precio = this.iva()+ this.valor;
precio = precio - precio*reb/100;
return Math.round(precio);
}
function obj_articulo(desc, cod, precio)
{
this.desc = desc;
this.codigo = cod;
```

```
this.valor = precio;
this.iva = iva;
this.pvp = total;
}
// -->
</SCRIPT>
```

Sección **Body**

```
<SCRIPT TYPE="text/javascript" LANGUAGE="JavaScript">
<!--
var item = new obj_articulo("Raton PS2", "PerRt-01", 500);
document.write('Artículo: ' + item.desc + " ("+item.codigo+")"+ "
<br>");
document.write("Precio: " +item.valor + " Ptas<br>");
document.write("IVA: " + item.iva() + " Ptas<br>");
document.write("Precio venta: "+ item.pvp(10)+ " Ptas (Dto 10%)<br>");
// -->
</SCRIPT>
```

Comentarios

Este ejemplo ilustra como crear un **Objeto** con sus propiedades y métodos. La función **obj_articulo** es el constructor del objeto mientras que las funciones **iva()** y **total(reb)** son métodos para ese objeto. El argumento de la segunda indica un posible descuento. Estas funciones son simples operaciones aritméticas y no requieren mas explicación. En cuanto al constructor las propiedades (**desc**, **codigo**, **valor**) se asignan directamente usando **this** para referirnos al propio objeto. Para los métodos se hace exactamente igual pero asignándole funciones (**iva**, **total**). Tenemos un método que usa argumentos y otro sin ellos, es decir este ejemplo presenta todas las posibilidades en creación de objetos y podría ser la base para crear una lista de la compra una de cuyas propiedades serían un objeto del tipo aquí definido.

7) Aplicaciones JavaScript HTML:

Aquí tienes algunos scripts útiles para aplicarlos directamente a tus páginas web. Son ejemplos de HTML dinámico donde se manejan los objetos que el explorador expone al JavaScript. El código de cada script está en la propia página, en la sección HEAD. Si quieres usarlo sólo debes seleccionarlo con el ratón y copiarlo con CTRL+C o con el menú del botón derecho, y luego lo pegas en la página donde quieras usarlo.

Ejemplos

A) Reloj en Pantalla	**K) Password 1**
B) Fecha de Actualización	**L) Título de página animado**
C) Menús Desplegables (IE)	**M) Bloque fijo**
D) Formularios de Correo	**N) Paisaje Nevado 1**
E) Personalizar Colores	**O) Paisaje Nevado 2**
F) Persianas	**P) Estrella Navideña 1**
G) Rollover	**Q) Estrella Navideña 2**
H) Información del Navegador	**R) Buscador en la Página**
I) Esquema desplegable (IE)	**S) Página de Inicio**
J) Botón más/menos	**T) Carrusel de Imágenes**

A) Reloj en Pantalla

a) Código JavaScript

Primero comprobamos el navegador: si es Netscape o Microsoft

```
var mie =(navigator.appName.indexOf("Microsoft")>=0)

function actReloj()
{
var hhmmss=new Date()
var horas=hhmmss.getHours()
var minutos=hhmmss.getMinutes()
var segundos=hhmmss.getSeconds()
```

Convertimos los números a dos dígitos, o sea, 6 -> 06

```
horas = (horas<=9)?("0"+horas):horas;
minutos = (minutos<=9)?("0"+minutos):minutos;
segundos=(segundos<=9)? ("0"+segundos):segundos;
```

Aquí construimos la cadena de texto HTML con la hora, y el tipo de letra que se escribe en la capa del reloj.

```
reloj="<font size='5' face='Arial' ><b>"+horas+":"+minutos+":"
+segundos +"</b></font>"
if (mie)
    {
     reloj.innerHTML=reloj;
    }
else
    {
      document.layers.reloj.document.write(reloj);
      document.layers.reloj.document.close();
    }
//ejecuta la función cada segundo (1000 miliseg)
setTimeout("actReloj()",1000);
}
```

b) Código HTML

```
<HTML>
<HEAD>
<SCRIPT language="JavaScript">
<!---Aquí sitúas el código javascript --->
</script>
</HEAD>
<!-- El reloj se lanza tras cargarse la página -->
<BODY onload="actReloj()">
<!-- La etiqueta Div que sigue define la capa que contendrá el reloj y
su posición vertical (top) y horizontal (left) -->
<div id="reloj"
style="position:absolute; top: 100; left: 100"></div>

</BODY>
</HTML>
```

B**)** Fecha de Actualización

a) Código JavaScript

Primero leemos en **cfecha** una cadena con la fecha correspondiente a la última vez que se modificó el documento. Convertimos **cfecha** en un objeto tipo **Date** que llamamos **fecha**. De este objeto extraemos el día de mes, el número de mes y el año. Para que el mes salga con su nombre simplemente usamos un **array** con los nombres de los meses ordenados, de manera que el número de mes extraído del objeto **fecha** nos sirve como índice en el array **meses** para averiguar el nombre del mes.

```
function ultActual()
{
var cfecha = document.lastModified;
var meses = new Array("Enero", "Febrero","Marzo","Abril",
"Mayo","Junio", "Julio", "Agosto", "Septiembre", "Octubre",
"Noviembre", "Diciembre");
var fecha = new Date(Date.parse(cfecha));
var dia = fecha.getDate();
var mes = meses[fecha.getMonth()];
var agno = fecha.getYear();
return (dia +" de "+mes +" de "+agno);
}
```

b) Código HTML

La llamada a la función **ultActual()** se coloca en la parte de la página que queramos, y por supuesto podemos ponerle los atributos o el estilo que deseemos.

```
<HTML>
<HEAD>
<SCRIPT language="JavaScript">
<!---Aquí sitúas el código javascript --->
</script>
</HEAD>
<BODY>
La página actual se actualizó el
<script language="JavaScript">
document.write("Esta página se actualizó el "+ultActual());
</script>
</BODY>
</HTML>
```

C) Menús Desplegables (IE)

a) Código JavaScript

La primera función, **despMenu()**, sirve para mostrar u ocultar la capa que contiene los items del menú, el primer argumento es el ID de la capa y el segundo es 0 para ocultar y 1 para mostrar. La segunda función, **destacar(),**se usa para resaltar modificar el color de fondo del elemento del menú cando pase el ratón, el primer argumento es el ID de la capa que contiene al elemento del menú y el segundo se interpreta como sigue:
0: indica que se debe resaltar, el ratón está encima.
1: se vuelve a la normalidad. El color original se guarda en la variable global itemOrig.

```
<script language="JavaScript">
<!--
```

Primero comprobamos el navegador: si es Netscape o Microsoft

```
var mie =(navigator.appName.indexOf("Microsoft")>=0)

var itemOrig;
function despMenu(nombre,sn)
{
obj = document.all[nombre];
if (sn>0)
   obj.style.visibility = "visible";
else
   obj.style.visibility = "hidden";
}
function destacar(obj, val)
{
if (val==1)
  {
   itemOrig = obj.style.backgroundColor;
   obj.style.backgroundColor="Aqua";
   }
else
   obj.style.backgroundColor= itemOrig;
}
-->
</script>
```

b) Código HTML

Puedes personalizar este ejemplo modificando los estilos de cabecera, items y capa del menú, o el atributo a cambiar en los elementos del menú, etc. A tu gusto.

```
<HTML>
<HEAD>
<!-- Definimos estilos para el menu -->
<style type="text/css">
<!--
.itMenu { position: absolute; clip: rect( ); background: #99FF99;
visibility: hidden}
.cabMenu { position: absolute; clip: rect( ); color: #FFFF66;
background: #0033FF}
.itMenuAct { background: #CCFFFF; width: 160px }
.itMenuDes { background: #99FF99; width: 160px }
-->
</style>
<SCRIPT language="JavaScript">
<!---Aquí sitúas el código javascript --->
```

```
</script>
</HEAD>

<BODY>
<!-- La etiqueta DIV que sigue define la capa con la cabecera del
menú. Cuando el ratón pase por aquí se despliega el menú-->
<div id="Menu1" style="width:140px; height:21px; z-index:2;
left: 39px; top: 23px" class="cabMenu"
onMouseOver="despMenu('itMenu1',1)"
onMouseout="despMenu('itMenu1',0)">Cabecera del menú
</div>

<!-- La etiqueta DIV que sigue define la capa que contendrá el menú,
la que se despliega cuando el ratón pase por la cabecera-->

<div id="itMenu1" style="width:103px; height:75px; z-index:1; left:
39px; top: 44px" class="itMenu"
onMouseOver="despMenu(this.id,1)" onMouseout="despMenu(this.id,0)">
<div id="itMenu11" class="itMenuDes" onMouseover="destacar(this,1)"
onMouseout="destacar(this,0)" >
<a href="destino1.htm" >Item 1</a>
</div>
<div id="itMenu12" class="itMenuDes" onMouseover="destacar(this,1)"
onMouseout="destacar(this,0)" >
<a href="destino1.htm" >Item 2</a>
</div>
<div id="itMenu13" class="itMenuDes" onMouseover="destacar(this,1)"
onMouseout="destacar(this,0)" >
<a href="destino1.htm" >Item 3</a>
</div>
<div id="itMenu14" class="itMenuDes" onMouseover="destacar(this,1)"
onMouseout="destacar(this,0)" >
<a href="destino1.htm" >Item 4</a>
</div>
</div>
Resto de la página web
</BODY>
</HTML>
```

D) Formulario de Correo

a) Código JavaScript

Aquí tienes una aplicación para crear tus propios formularios con envío de los datos por correo. El código Javascript es de lo más simple. Esta es una muestra que deberás personalizar para tus propias necesidades. Como siempre puedes usarlo libremente, a cambio te agradecería que colocaras un link a este sitio.
Esta función se usa para validar el formulario, aunque también podría hacer otras cosas. Es llamada por el evento onsubmit del formulario, como puedes ver en el código HTML.
En este caso concreto si en el formulario no se ha seleccionado al menos una de las opciones la función devuelve falso y el formulario no se envía, en caso contrario todo sigue su curso y el formulario se envía. La función recibe como argumento el objeto formulario y los valores de sus campos se identifican de la siguiente forma: **obj.name_en_el_campo.value**, salvo en el caso de los **checkbox** y **radiobutton** que se identifican como elementos de un **array** seguidos de la propiedad **checked** que es true si está marcado o false si no lo está. En el cuerpo de los mensajes enviados con este

formulario recibirás en tu correo los valores de cada campo como pares campo=valor_del_campo. Pruébalo y verás que simple resulta.

```
function Comprobar(obj)
{
if (!obj.Opciones[0].checked &&
    !obj.Opciones[1].checked &&
    !obj.Opciones[2].checked)
    {
     alert("Debe elegir una opción")
     return false;
     }
else
     return true
}
```

b) Código HTML

Aquí tienes el código HTML para poner en la sección <**body**> de la página donde coloques el formulario. Ya sabes: seleccionar, copiar y pegar. Personalízalo a tu gusto. Contiene campos de texto, checkbox, radiobutton y menus desplegables. En el campo **action** colocas la dirección e-mail donde quieras recibir el formulario, el campo ?subject=asunto es opcional, pero muy útil a la hora de filtrar la respuesta y enviarla a una carpeta determinada con tu programa de correo. La clave de esta aplicación está en el evento **onsubmit**, si la función colocada en ese evento devuelve true el formulario se envía y si devuelve **false** no se envía. Así de fácil.

```
<form name="form1" method="POST" onsubmit="return Comprobar(this)"
action= "mailto:poneaquí@tuemail?subject=asunto" enctype ="text/plain"
>
<p>Campo de texto
<input type="text" name="textfield" size="80">
<br>
Lista de selección
<select name="select">
<option value="Valor 1" selected>Elemento 1</option>
<option value="Valor 2">Elemento 2</option>
<option value="Valor 3">Elemento 3</option>
</select>
<br>
Campos excluyentes
<input type="radio" name="radio" value="Botón 1">
<input type="radio" name="radio" value="Botón 2">
<br>
Campos para opciones: <br>
Opción 1
<input type="checkbox" name="Opciones" value="opcion 1"><br>
Opción 2
<input type="checkbox" name="Opciones" value="opcion 2"><br>
Opción 3
<input type="checkbox" name="Opciones" value="opcion 3"></p>
<p align="center">
<input type="reset" name="Reset" value="Borrar">
<input type="submit" name="Submit" value="Enviar">
</p>
</form>
```

E) Personalizar Colores

a) Código JavaScript

Mediante este sencillo código los usuarios pueden seleccionar la combinación de colores de la página. El código del script es muy simple: tan sólo asigna los colores e fondo y texto elegidos por el usuario a las propiedades **bgColor** y **fgColor** del **document**. Los valores son elegidos por el usuario a partir de una lista desplegable. Este código está pensado para IExplorer, pues Netscape no permite modificar estos argumentos cuando la página ya esté cargada. Como es habitual sitúa este código en la sección Head de la página.

```
<script language="JavaScript">
<!--
```

Primero comprobamos el navegador: Si es Netscape o Microsoft

```
var mie =(navigator.appName.indexOf("Microsoft")>=0)

function cambColores(fondos, letras)
{
 var ind;
 if (!mie) return
 ind = fondos.selectedIndex;
 document.bgColor = fondos.options[ind].value;
 ind = letras.selectedIndex;
 document.fgColor = letras.options[ind].value;
 return;
}
-->
```

b) Código HTML

La primera línea, con formato H2, es tan sólo para muestra. El formulario posee dos campos con las listas de los colores seleccionables por el visitante. Al picar con el ratón sobre el botón se dispara el evento que pone en marcha la función **cambColores**. Si quieres mas colores tan sólo tienes que añadir opciones al menú. El color puede ponerse por su nombre como en el ejemplo o mediante las cadenas RGB (000000 a FFFFFF).

```
<h2>Esta es una muestra de personalización de colores</h2>
<form name="form1" method="post" action="">
Fondo:
<select name="fondo">
<option value="yellow">Amarillo</option>
<option value="white">blanco</option>
<option value="Blue">Azul</option>
</select>
Texto:
<select name="texto">
<option value="yellow">Amarillo</option>
<option value="white">blanco</option>
<option value="Blue">Azul</option>
</select>
<input type="button" name="Button" value="cambiar"
onclick ="cambColores(parentElement.fondo,parentElement.texto)">
</form>
```

F) Persianas

a) Código JavaScript

Un código aparentemente complejo pero que responde a una idea muy simple: Ir aumentando la altura de la región de recorte de la capa que contiene al párrafo móvil, usando para ello un evento temporizador. Pero claro si sólo hacemos esto iríamos viendo como el párrafo va siendo descubierto pero sin sensación de movimiento. Es como si tapamos un texto con una hoja de papel y la vamos bajando poco a poco. ¿Como lograr que el párrafo se mueva? Centrémonos en el scroll de abajo hacia arriba: usamos una región de recorte (CLIP) inicial estrecha, sólo se ve la primera línea; luego aumentamos el tamaño de esa región bajando su límite inferior y llevamos hacia arriba la posición vertical de la capa. Repetimos esto hasta que toda la capa sea visible.

Primero comprobamos el navegador: Si es Netscape o Microsoft

```
var mie =(navigator.appName.indexOf("Microsoft")>=0)
var txtAct="";
```

Argumentos:
primera: true para la primera vez que se ejecute
capa: id del objeto DIV con el texto a desplazar
sent: 0 de arriba-abajo, 1: de abajo-arriba
vel: rapidez del desplazamiento (0 a 100)

```
function vertical(primera, capa, sent, vel)
{
var dimen=0;
//Primera vez que se ejecuta, iniciar todo
if (primera)
   {
   //txtAct: objeto global con la capa del scroll.
   txtAct = (mie)?document.all[capa]:document.layers[capa];
   txtAct.alto = (mie)?txtAct.offsetHeight:txtAct.clip.height;
   txtAct.clp = 0;
   if (mie)
      txtAct.sup = txtAct.style.posTop+txtAct.alto*sent
   else
      txtAct.sup = txtAct.top+txtAct.alto*sent;
   txtAct.incr = Math.round(txtAct.alto*vel/100);
   }
else
   {
   txtAct.clp += txtAct.incr;
//La región de recorte no puede tener una altura mayor del 100%
   if (txtAct.clp > 100)
       txtAct.clp = 100;
   }
if (sent>0)
   dimen = txtAct.clp;
else
   dimen = 100 - txtAct.clp;
if (mie)
   {
    if (sent>0) //de Abajo hacia Arriba sent=1
      {
      txtAct.style.clip = 'rect(auto, auto,'+ dimen+'%, auto)'
```

```
      }
   else
      {
      txtAct.style.clip = 'rect('+ dimen+'%, auto, auto, auto)'
      }
   txtAct.style.posTop=
          Math.round(txtAct.sup - txtAct.alto*dimen/100);
   }
else
   {
    if(sent>0)
        txtAct.clip.bottom = Math.round(dimen*txtAct.alto/100);
    else
        txtAct.clip.top = Math.round(dimen*txtAct.alto/100);
    txtAct.top = Math.round(txtAct.sup - txtAct.alto*dimen/100);
   }
if (primera)
    verCapa(txtAct, true);
if (txtAct.clp < 100)
    setTimeout("vertical(false,'',"+sent+","+txtAct.incr+")",
               txtAct.vel);
else
    txtAct.clp = -1;
}
//Muestra u oculta una capa
function verCapa(obj, sn)
{
var mostrar = (sn)?'block':'none';
var estado = (sn)?'visible':'hidden';
if (mie)
   {obj.style.display = mostrar;
   obj.style.visibility= estado;
   }
else
   obj.visibility = estado;
}
-->
</script>
```

b) Código HTML

Puedes usar esta página para comprobar el funcionamiento de la función descrita en este script. La capa con el texto aparecerá de abajo hacia arriba si pulsas el botón **Arriba** y en sentido inverso si pulsas el botón **Abajo**.

```
<HMTL>
<HEAD>
<!-- Definimos estilos para el menu -->
<SCRIPT language="JavaScript">
<!---Aquí sitúas el código javascript --->
</script>
</HEAD>
<BODY>
<div id="Layer1" style="position:absolute; width:200px; height:115px;
z-index:1; left: 200px; top: 138px; background: #CCFF66; layer-
background-color: #CCFF66; border: 1px none #000000; visibility:
hidden">
Este es el texto que queremos ver desplazándose y con los botones de
arriba podemos hacer que el párrafo vaya apareciendo de arriba a bajo
o de abajo arriba. Existe otra forma de lograr este efecto usando
capas anidadas, pero Netscape se lleva mal con los anidamientos de
```

```
capas.</div>
<input type="button" name="Button" value="Arriba"
onclick="vertical(true,'Layer1', 1, 2)">
<input type="button" name="Button" value="Abajo"
onclick="vertical(true,'Layer1', 0,2)">
</BODY>
</HTML>
```

G) Rollover

a) Código JavaScript

Con este ejemplo, te podrás hacer que tus enlaces cambien al pasar el ratón por encima. Es muy sencillo y podrás modificarlo para lograr efectos vistosos.
Esta función primero comprueba el navegador, pues MIE usa la propiedad **style** del objeto para acceder a los atributos de estilo del objeto, mientras que NS accede directamente. Otra diferencia es que MIE admite cualquier propiedad mientras que NS4 sólo puede cambiar el color de fondo.
Si usas el objeto navegador descrito en otro de los scripts de estos ejemplos puedes cambiar la variable mie por algo así como visita.IE (si visita es el objeto oNavegador).

Primero comprobamos el navegador: si es Netscape o Microsoft. Si usas el objeto navegador esta instrucción podría no ser necesaria

```
var mie =(navigator.appName.indexOf("Microsoft")>=0)

function cambProp(obj, propIE, propNS, valorIE, valorNS)
{
if (mie)
   {
   obj = document.all[obj];
   obj.style[propIE] = valorIE;
   }
else
   {
   obj = document.layers[obj];
   obj[propNS] = valorNS;
   }
}
```

b) Código HTML

Este sencillo ejemplo te permite cambiar el color de fondo del enlace al pasar el ratón por encima. Funciona tanto en MIE como en NS4, de ahí la necesidad del **span** y de tantos atributos. Aquí se ha usado un nombre de color, pero puedes usar los códigos hexadecimales en su lugar como #FF00AA o el que te parezca. En el evento **onmouseout** el color de fondo para MIE es una cadena vacía que se interpreta como transparente, mientras que NS usa **null** para lograr el mismo efecto. Recuerda que MIE admite cualquier cambio en los atributos del estilo, mientras que NS sólo permite cambiar el color y la imagen de fondo, al menos hasta la versión 4.7.

```
<HTML>
<HEAD>
<!-- aquí va el código javascrip -->
</HEAD>
<BODY color ="#FFFFFF">
```

```
<A HREF="pagina.htm"
onmouseover="cambProp('enlace', 'background', 'bgColor',
'yellow','yellow')"
onmouseout="cambProp('enlace', 'background','bgColor','',null)">
<span id='enlace' style="position:relative">
rollover
</span>
</A>
</BODY>
</HTML>
```

H) Información del Navegador

Este nuevo script te permitirá conocer que Navegador están usando tus visitantes, y así podrás presentar tu página de forma más adecuada.

a) Código JavaScript

Como ves en este script usamos un objeto llamado **o_Navegador** con varias propiedades y un método:

	Propiedades
nombre	string, guarda el nombre del navegador.
version	number, número completo de la versión (5.01, 6.02)
Verent	number, número de versión base (4, 5, 6, etc.)
standard	lógico, verdadero si el navegador cumple los standards de W3C
IE	lógico, verdadero para el navegador de Microsoft
NS	lógico, verdadero para el navegador de Netscape
OP	lógico, verdadero para el navegador Opera
XX	lógico, verdadero si es otro navegador.
	Métodos
iniciar	averigua y coloca adecuadamente los valores de las propiedades de la instancia del objeto. Este método es usado por el constructor del objeto, la función con el nombre oNavegador( )

Así pues queda como:

```
function o_Navegador() {
this.nombre = navigator.appName;
this.iniciar = iniciar;
this.IE = this.nombre.toUpperCase().indexOf('MICROSOFT') >=0;
this.NS = this.nombre.toUpperCase().indexOf('NETSCAPE') >=0;
this.OP = this.nombre.toUpperCase().indexOf('OPERA') >= 0;
this.XX = !this.IE && !this.NS && !this.OP;
this.version = this.iniciar();
this.Verent = parseInt(this.version);
this.standard = (this.IE && this.Verent >=6) || (this.NS &&
this.Verent >=6)
/* ==================================================================
FUNCION: iniciar
ARGS: ninguno.
RETURN: nada
DESCR: Inicializa los valores del objeto
================================================================== */
 function iniciar() {
```

```
 var ver = navigator.appVersion;
 if(ver+"" != "NaN")
 if (this.IE)
   {
   ver.match(/(MSIE)(\s*)([0-9].[0-9]+)/ig);
   ver = RegExp.$3;
   }
  return ver;
  } //Termina la funcion iniciar el objeto
}//Termina la definición del objeto
```

b) Código HTML

Esta es una muestra muy simple del uso del objeto oNavegador, tan sólo para comprobar que funciona y mostrar su uso. Como observarás de entrada es necesario definir el nombre del objeto que vayas a usar para almacenar la información del navegador y luego usas este objeto convenientemente dentro de tus scripts.
Si vas a usar este objeto crea una instancia única definida en un archivo .js (lo lógico es que sea el mismo que contenga la definición del objeto **oNavegador**) que enlazarás en todas las páginas en las que uses scripts. Cada script que dependa del navegador consultará este objeto. De esta forma tienes un objeto normalizado para tus páginas y cuando aparezcan nuevos navegadores sólo tienes que modificar este fichero.

```
<body>
<script language="Javascript">
visita = new o_Navegador();
document.write("Navegador: "+visita.nombre+"<br>");
document.write("  Versión: "+visita.version+"<br>");
if (visita.standard)
   {
   document.write("Este navegador cumple con la regulación ");
   document.write("DOM1 del W3C<br>");
   }
else
   {
   document.write("Este navegador no cumple la regulación ");
   document.write("DOM1 del W3C<br>");
   }
</script>
</body>
```

I**)** Esquema desplegable (IE)

Este pequeño ejemplo te permitirá construir listas en las que los niveles se pueden abrir y cerrar a voluntad. Pruébalo y personalízalo para tus necesidades.

a) Código JavaScript

Se trata de un script tan simple que apenas necesita comentarios. La función recibe como argumentos un identificador (iden), y busca el elemento HTML cuyo ID sea igual a este valor, para ello usa el método getElementById( ). Una vez encontrado el elemento tan sólo modifica su propiedad **display**: si está en mostrar (block) lo pone en ocultar (none) y viceversa.
Quizás lo más interesante de este script sea que usa código standard, es decir destinado a navegadores que cumplen con los estándares de la W3C, que en lo que afecta al DHTML es la normativa del DOM1 y el HTML4. Por tanto este script debe funcionar

en Netscape 6 y en MSIE 6, de hecho funciona a partir de MSIE 5, que sólo cumple la regulación DOM1 en un 86%.

```
<script language="JavaScript">
function cambiaEstado(iden)
{
var elhtml = document.getElementById(iden);
if (elhtml.style.display == 'block')
   elhtml.style.display = 'none';
else
   elhtml.style.display = 'block';
}
</script>
```

b) Código HTML

Como vemos en lugar de usar el evento onclick se hace la llamada a la función cambiaEstado( ) directamente en el enlace. El argumento de la llamada es el ID del bloque UL de la lista. Simple.

```
<body>
<ul>
<li><a href="javascript:cambiaEstado('e1')">entrada 1</a>
<ul id="e1" class="nivel1">
<li>apartado 1 de entrada 1</li>
<li>apartado 2 de entrada 1</li>
</ul>
</li>
<li><a href="javascript:cambiaEstado('e2')">entrada 2</a>
<ul id="e2" class="nivel1">
<li>apartado 1 de entrada 2</li>
<li>apartado 2 de entrada 2 </li>
</ul>
</li>
</ul>
</body>
```

J) Botón más/menos

Es una forma de añadir a tus formularios, los botones de control de tipo Windows, que evitan tener que escribir los números en los cuadros de diálogo.

a) Código JavaScript

Este es un objeto usado para construir un botón de control para formularios HTML. Su misión consiste en incrementar o decrementar el contenido de un campo del formulario dependiendo de en que parte del botón pulsemos con el ratón. El objeto se denomina oBotonCtrl, sus propiedades y métodos son:

Propiedades	
control	object, El objeto campo del formulario sobre el que actúa
max	número, Valor máximo del contenido del campo
min	número, Valor mínimo del contenido del campo
seguir	-1, 0, 1. Es el valor en que se incrementa el valor del campo asociado
Métodos	
mas	Pone a 1 el valor de **seguir** y llama a la función **cambiar**

menos	Pone a -1 el valor de **seguir** y llama a la función **cambiar**
parar	Pone a o el valor de **seguir** y llama a la función **cambiar**
cambiar	Usa el valor de seguir para actualizar el contenido del campo asociado. Si el valor de seguir no es cero inicializa el temporizador para que vuelva a llamar a este método, con lo que el valor continuará modificándose.

El objeto se crea como es habitual con el operador new:
boton1 = new oBotonCtrl(formul, control, -20, 20)
donde formul es un identificador de un formulario y control es el nombre (name) del control sobre el que actuará el botón. El formulario debe existir cuando se cree el objeto botón.

```
<script language="javascript">
function oBotonCtrl(formul, idcampo, min, max)
{
this.control = "document.forms."+formul+"."+idcampo;
this.min = min;
this.max = max;
this.seguir = 0;
this.mas = new Function("this.seguir = 1; this.cambiar()");
this.menos = new Function("this.seguir = -1; this.cambiar()");
this.parar = new Function("this.seguir=0");
this.cambiar = cambiar;
function cambiar()
{
var lim = this.seguir>0?this.max:this.min;
//Crea la variable global $boton para el temporizador
if (!window.$boton)
    window.$boton = this;
var ctrl = eval(this.control);
if (ctrl.value == '')
    ctrl.value = 0;
if (ctrl.value != lim)
   {
   ctrl.value = parseInt(ctrl.value)+this.seguir;
   if(this.seguir!=0)
      //Llama al método cambiar del objeto actual
      setTimeout("$boton.cambiar()", 200);
   else
      window.$boton = null;
   }
 }
}//Fin del objeto botón

</script>
```

b) Código HTML

En el código HTML como puedes observar se utiliza un image map, o sea, una imagen con áreas activas que hacen de enlaces, la cual puedes crear con programas como Fireworks o tomados de otras webs.
Cuando la imagen se ha cargado crea el objeto **boton2** del tipo **oBotonCtrl**, que está asociado al campo control del formulario **form2**, siendo -50 y 50 los valores límites para el contenido del campo. El objeto también puede crearse mediante un evento **onload** del **body**. En el image map existen dos áreas, una para cada mitad del botón. Al pulsar el área superior se llama al método **mas()** del **boton2** usando un evento **onmousedown**, cuando soltamos el botón se genera un evento **onmouseup** que llama al

método **parar()** del **boton2**. Como reto queda adaptar este script para que además de números pueda manejar listas de valores. ¿Alguien se atreve? Quien lo logre que comparta su trabajo.

```
<BODY>
<FORM NAME="form2" METHOD="post" ACTION="">
<INPUT TYPE="text" name="control">
<MAP NAME="m_ctrl2">
<area shape="rect" coords="0,0,13,10" href="javascript:void(0)"
onmousedown="boton2.mas()" onmouseup="boton2.parar()">
<area shape="rect" coords="0,11,13,20" href="javascript:void(0)"
onmousedown="boton2.menos()" onmouseup="boton2.parar()">
</MAP>
<img name="ctrl2" src="ctrl.gif" width="13" height="20" border="0"
usemap="#m_ctrl2" align="absmiddle" hspace="0"
onLoad="window.boton2 = new oBotonCtrl('form2','control', -50, 50)">
</FORM>
</BODY>
```

K) Password 1

Una sencilla forma de acceder a una página, mediante un password o contraseña, empleando para ello JavaScript.

a) Código JavaScript

La primera función estará en la página de acceso mientras que la sentencia para redireccionar se sitúa en la cabecera de la página a la que se accede desde la primera (página clave).

El método se basa en colocar páginas cuyo nombre sea el de las passwords, seguido por supuesto de la extensión htm o html. Cuando un usuario accede deberá conocer el nombre de la página si no recibirá un mensaje de página no encontrada emitido por el navegador. Al acceder a la página clave el navegador se redirecciona a la página donde tu quieras que llegue.

```
function password(texto)
{
if (texto!="")
   location.href= texto+".htm";
}

//sentencia para redireccionar
location.replace("paginabase.htm")
```

b) Código HTML

La página de acceso contendrá la función password() o mejor aún un enlace al archivo con esta función. La página clave se llamará con el nombre que asignes a la password, por ejemplo re345ty seguido de la extensión htm.

Cuando el usuario accede a tu página de acceso escribe en el formulario su clave, si ésta es re345ty llegará a la página re345ty.htm donde una sentencia de redirección lo envía a tu página principal. Si el usuario escribe otra clave, por ejemplo er456, el navegador

buscará la página correspondiente, en el ejemplo er456.htm, al no hallarla devolverá la clásica página de error. Deberás crear una página clave para cada password que asignes.

Sencillo ¿verdad?, no es a prueba de bombas pero puede funcionar y de hecho funciona. La forma de romper este password es encontrar la lista de directorios de tu web, y la mayoría de servidores no permiten esta operación. Claro también podrían acceder a esta página principal directamente si conocen su nombre.

Página de acceso:

```
<HTML>
<HEAD>
<!-- aquí va el código javascrip -->
</HEAD>
<BODY COLOR ="#FFFFFF">
<FORM NAME="pass" ACTION="">Teclea tu password:
<input type="password" name="clave">
<input type="button" name="entrar"
onclick = "password(this.form.clave.value)">
</FORM>
</BODY>
</HTML>
```

Página clave:

```
<HTML>
<HEAD>
<script language="javascript">
location.replace("mipaginappal.htm");
</script>
</HEAD>
<BODY COLOR ="#FFFFFF">
</BODY>
</HTML>
```

L**)** Título de página animado

Original Script, con el que entre otras cosas, animar el título de la página web o cambiarlo dinámicamente; válido para Internet Explorer 5 y Netscape 6.

a) Código JavaScript

Este sencillo script es una muestra de la aplicación del método global **setTimeout()**, un temporizador que aplica un retraso a la ejecución de una función. En este caso la función a la que se aplica es la denominada **animaTitulo( )** que se encarga de desplazar el título de la página creando un efecto marquesina.

La función se ejecuta cada 200 milisegundos, cuanto menor sea este número más rápido será el desplazamiento del título. Esto se consigue cogiendo el título en un momento dado y partiéndolo en dos partes: una va desde la segunda letra hasta el final, instrucción **substr(1)**, mientras que la segunda parte contiene el primer carácter, **substring(0, 1)**.

La variable **sp,** se utiliza porque si en un instante dado la cadena del título termina en un espacio, este espacio es desechado por MSIE y en la siguiente vuelta la letra siguiente al

espacio queda pegada a la anterior, así que en esta variable guardamos un espacio que se inserta entre ambos caracteres. Netscape 6 no presenta este fallo. El efecto puedes verlo en esta misma página.
De la misma forma que se crea este efecto con el título de la página podríamos ir cambiándolo entre distintos valores, o hacerlo guiñar.

```
var esp = '';
function animaTitulo()
{
var actual = document.title;
document.title = actual.substr(1) + sp + actual.substring(0, 1);
if (navigator.appVersion.indexOf('MSIE')>=0)
    sp = (actual.substring(0, 1)==' ')?" ":"";
setTimeout('animaTitulo()', 200);
}
```

b) Código HTML

El uso de este script es de lo más simple, basta vincular la función **animaTitulo( )** al evento **onload** de la página, y por supuesto ponerle un título. Si te fijas en el ejemplo, comprobarás que el título acaba con el carácter extendido   el cual se usa en HTML para indicar un espacio, esto tan sólo es necesario para que el principio y el fin de la cadena no queden pegados.

```
<HTML>
<HEAD>
<TITLE>Título animado de tu página.  </TITLE>
<!-- aquí va el código javascrip -->
</HEAD>
<body color ="#FFFFFF" onload="animaTitulo()">
<!--Contenido de tu página-->

</BODY>
</HTML>
```

M**)** Bloque fijo

a) Código JavaScript

Este sencillo código te permitirá colocar un bloque en tu página que permanecerá estático aunque hagas un scroll vertical en la ventana del navegador. La función tiene dos argumentos: **obj** y **posic**. El primero de ellos es el identificador del bloque que quieres clavar en la ventana, y el segundo es la posición vertical que quieres que ocupe. La función actúa ligada al evento **onscroll** del elemento **body** de la página, cuando se hace un scroll vertical esta función modifica la posición vertical del cuadro sumándole a la posición inicial el valor del desplazamiento vertical de la ventana, de esta forma el bloque parece permanecer fijo.

```
function clavar(obj, posic)
{
dObj =document.getElementById(obj);
dObj.style.posTop = document.body.scrollTop+posic;
}
```

Funciona a partir de IE5, si quieres que funcione en IE4 sustituye la primera línea con

```
dObj = document.all[obj];
```

El código del ejemplo tal y como está es compatible con el estándar W3C, algo muy recomendable ya que esta normativa parece ser la que está fijando el estándar para la programación en la red.

b) Código HTML

Para usar este script basta con vincular la función clavar al evento onscroll del elemento body. En la página deberá existir un bloque DIV con posicionamiento absoluto, cuyo atributo ID y posición inicial se usan como argumentos para la llamada a la función clavar. Este bloque puede tener el estilo definido en línea como en este ejemplo o bien definido como un estilo CSS, que en general es lo más aconsejable. En cuanto al contenido puede ser cualquier texto HTML, es decir, texto, imágenes, listas, un menú de enlaces, etc.

```
<body onscroll=clavar('cuadro', 100)>
<div id="cuadro" style="position:absolute; top:100; left:50;
width:200; height:200">Este contenido permanece fijo en la
ventana</div>
```

N**)** Paisaje Nevado 1

Si quieres adornar tus páginas Javascript, con este ejemplo podrás colocar una nevada en tu página

a) Código JavaScript

Este es un sencillo script para añadir un efecto de nevada en una página o en una zona de una página. En este ejemplo se usa el carácter * (asterisco) para representar los copos, lo que ahorra tiempo, pero puedes sustituirlo por una imagen de un copo de nieve si te apetece. Para ello en lugar de
document.write('*');
Debes poner
document.write('<img src="copo.gif">');
donde copo.gif será la dirección de imagen que hayas elegido.
El funcionamiento es muy simple: la función **iniCopos()** se encarga de situar sendos bloques DIV con los copos. A esta función se la llama con 6 argumentos: número de copos, ancho y alto de la zona nevada, origen x e y de la zona nevada y tiempo de caída. La función **iniNevada()** construye un array con los objetos DIV de los copos. La función nevar() se llama cíclicamente mediante un temporizador, y se encarga de ir haciendo bajar los copos.
Partiendo de este código básico seguro que puedes lograr efectos interesantes.

```
<script language="JavaScript">
function iniCopos(num, anc, alto, veloc)
{
var i, x, y;
for (i = 0; i<num; i++)
      {
      x = parseInt(Math.random()*anc);
      y = parseInt(Math.random()*alto);
      dibujaCopo(i,x,y);
      }
window.copos = new Array(num);
```

```
window.coposAlto = alto;
window.coposAncho = anc;
window.coposVeloc = veloc;
}
function dibujaCopo(num, x, y)
{
var tama = Math.round(Math.random())+ 3
document.write('<div id="c'+num+'" style="position:absolute;
width:1px;')
document.write('height:1px; z-index:0; left:'+x+'px; top:'+y+'"px; ')
document.write(' background-color: #FFFFFF">);
document.write('<font size="'+tama+'" color = "#FFFFFF" >*</font>');
document.write('</div>');
}
function iniNevada()
{
for (i = 0; i < window.copos.length; i++)
      if (document.all)
            window.copos[i] = document.all["c"+i]
      else
            window.copos[i] = document.getElementById("c"+i);
setTimeout("nevar()",1);
}
function nevar()
{
var x, y;
for (i = 0; i < copos.length; i++)
{
y = parseInt(window.copos[i].style.top);
x = parseInt(window.copos[i].style.left);
if (Math.random() > 0.5)
     y += parseInt(Math.random()+1);
y += parseInt(Math.random()+2);
window.copos[i].style.top = y;
if (y > document.body.scrollTop + coposAlto)
  {
  window.copos[i].style.top = document.body.scrollTop-
  Math.round(Math.random()*10);
  window.copos[i].style.left =
        parseInt(Math.random()*coposAncho-1);
  }
}
setTimeout("nevar()", coposVeloc);
}
</script>
```

b) Código HTML

El código de la página es bien simple: una llamada a la función iniCopos() en el cuerpo de la página dibujará los bloques DIV con los copos, mientras que un evento onload vinculado a la función iniNevada() hará que los copos comiencen a caer por el paisaje nevado.
Como siempre las funciones descritas en la página del código debes situarlas en la sección HEAD de la página.

```
<body onload="iniNevada()">
<script language="JavaScript">
iniCopos(50, 400, 300, 100, 100, 50);
</script>
</body>
```

O) Paisaje Nevado 2

Otro script para colocar una nevada navideña en tu página. Se trata de un script bastante completo donde puedes aprender un montón de cositas sobre los objetos Javascript.

a) Código JavaScript

Este es un sencillo script para añadir un efecto de nevada en una página o en una zona de una página. En este ejemplo se usa el carácter * (asterisco) para representar los copos, lo que ahorra tiempo, pero puedes sustituirlo por una imagen de un copo de nieve si te apetece. Para ello en lugar de
document.write('*');
Debes poner
document.write('<img src="copo.gif">');
donde copo.gif será la dirección de imagen que hayas elegido.
El funcionamiento es muy simple: la función **iniCopos()** se encarga de situar sendos bloques DIV con los copos. A esta función se la llama con 6 argumentos: número de copos, ancho y alto de la zona nevada, origen x e y de la zona nevada y tiempo de caída. La función **iniNevada()** construye un array con los objetos DIV de los copos. La función nevar() se llama cíclicamente mediante un temporizador, y se encarga de ir haciendo bajar los copos.
Partiendo de este código básico seguro que puedes lograr efectos interesantes.

```
<script language="JavaScript">
function iniCopos(num, anc, alto, veloc)
{
var i, x, y;
for (i = 0; i<num; i++)
      {
      x = parseInt(Math.random()*anc);
      y = parseInt(Math.random()*alto);
      dibujaCopo(i,x,y);
      }
window.copos = new Array(num);
window.coposAlto = alto;
window.coposAncho = anc;
window.coposVeloc = veloc;
}
function dibujaCopo(num, x, y)
{
var tama = Math.round(Math.random())+ 3
document.write('<div id="c'+num+'" style="position:absolute;
width:1px;')
document.write('height:1px; z-index:0; left:'+x+'px; top:'+y+'"px; ')
document.write(' background-color: #FFFFFF">);
document.write('<font size="'+tama+'" color = "#FFFFFF" >*</font>');
document.write('</div>');
}
function iniNevada()
{
for (i = 0; i < window.copos.length; i++)
      if (document.all)
            window.copos[i] = document.all["c"+i]
      else
```

```
            window.copos[i] = document.getElementById("c"+i);
setTimeout("nevar()",1);
}
function nevar()
{
var x, y;
for (i = 0; i < copos.length; i++)
{
y = parseInt(window.copos[i].style.top);
x = parseInt(window.copos[i].style.left);
if (Math.random() > 0.5)
     y += parseInt(Math.random()+1);
y += parseInt(Math.random()+2);
window.copos[i].style.top = y;
if (y > document.body.scrollTop + coposAlto)
  {
  window.copos[i].style.top = document.body.scrollTop-
  Math.round(Math.random()*10);
  window.copos[i].style.left =
        parseInt(Math.random()*coposAncho-1);
  }
}
setTimeout("nevar()", coposVeloc);
}
</script>
```

b) Código HTML

El código de la página es bien simple: una llamada a la función iniCopos() en el cuerpo de la página dibujará los bloques DIV con los copos, mientras que un evento onload vinculado a la función iniNevada() hará que los copos comiencen a caer por el paisaje nevado.
Como siempre las funciones descritas en la página del código debes situarlas en la sección HEAD de la página.

```
<body onload="iniNevada()">
<script language="JavaScript">
iniCopos(50, 400, 300, 100, 100, 50);
</script>
</body>
```

P**)** Estrella Navideña 1

¿Te gustaría ver la estrella de la Navidad en tu página? Sigue las instrucciones de este script y podrás ver como la estrella o Santa. Claus o los renos atraviesan tu página.

a) Código JavaScript

Este ejemplo utiliza los objetos móviles que se pueden encontrar en http://www.ciudadfutura.com/javascriptdesdecero/casos/movilenlaz/moviles.htm. La función que desencadena todo el movimiento es iniAnima(), en ella se define **estnav** como un objeto **oMovil( )**, los argumentos para crearlos son: ID de la capa DIV donde se encuentra la estrella, posiciones X, Y iniciales y finales del movimiento, **tiempo** que debe tardar ese movimiento, un **valor lógico** (true o false) para indicar si el objeto debe permanecer visible al acabar el movimiento y el **nombre de la función** que define el movimiento de la estrella, esta última función es optativa, si no se usa la estrella se

moverá linealmente desde la posición inicial a la final. En este ejemplo se ha usado una función senoidal, puedes usar cualquier otra. Si quieres que el movimiento se repita una y otra vez sólo tienes que añadir la sentencia:
animEst.enlazar('animEst',null,3000);
Antes de animEst.iniciar(). El argumento último (3000 en este caso) indica los milisegundos que tardará en repetirse el movimiento.

```
<script language="JavaScript" src="rutbasicas.js">
</script>
<script language="JavaScript" src="objmovilenl.js">
</script>
<script language="JavaScript">
     function iniAnima()
     {
     var estnav = new oMovil('estrella', 10, 200,600, 190, 60,
                              true, 'movEstrella');
     estnav.situar()
     animEst = new oAnimacion(estnav)
     animEst.iniciar('animEst');
     }
     function movEstrella()
     {
     var y, x;
     x = Math.round(this.leeX()+this.incx);
     y = this.py0 + Math.round(40*Math.sin(x*Math.PI/180))
     this.ponY(y);
     this.ponX(x);
     this.situar();
     this.esFinal = Math.abs(this.px1-x) < Math.abs(this.incx);
     }
</script>
```

b) Código HTML

En la página sólo debes colocar la llamad a la función que inicia todo el movimiento y por supuesto un bloque DIV para contener a la figura que desees mover, puede ser la estrella del ejemplo o un Papá Noel o los tres reyes magos o los renos ....

```
<html>
<head>
<title>
Animando el título de tu página.  </title>
<!-- aquí va el código javascrip -->
</head>
<body color ="#FFFFFF" onload="iniAnima()">

<DiV id="estrella" style="position:absolute; z-index:1; width: 1;
height: 1; overflow: visible; left: 6px; top: 14px"><img
src="../../simbolos/estnav.gif" width="57" height="37"></div>
<!--Contenido de tu página-->
</body>
</html>
```

Q) Estrella Navideña 2

Este texto muelle seguirá a tu ratón por toda la página con un efecto elástico muy vistoso. Aprovéchalo para colocar un mensaje de felicitación de navidad... o cualquier otro uso.

a) Código JavaScript

Aquí tienes un ejemplo de utilización del objeto ratón con estela que puedes descargar en la página:
http://www.ciudadfutura.com/javascriptdesdecero/casos/movilenlaz/moviles.htm

Se trata de una frase que seguirá al ratón con un movimiento elástico. Si observas el ratón en esta página comprenderás el efecto que se hace más evidente si mueves el ratón con cierta rapidez. Después de leer la documentación de este objeto te será fácil comprender el funcionamiento del script. Descarga en tu sitio el archivo js del objeto estela, o enlázalo al de esta web, y añade las funciones que aquí te presentamos. La primera de estas funciones es **iniciarEstela(texto)**, el argumento es el texto que seguirá al ratón. Esta función coloca en la página las letras de la frase que seguirá al ratón (cada letra se coloca en un bloque DIV para poder controlar su posición), además crea e inicializa el objeto oEstela propiamente dicho y vincula los eventos de movimiento del ratón y el temporizador para ir actualizando la posición del texto. La otra función, **mianimación()**, es la que se encarga de ir colocando las letras para que sigan al ratón con este curioso efecto.

```
<script language="JavaScript" src="rutbasicas.js">
</script>
<script language="JavaScript" src="objmovilenl.js">
</script>
<script language="JavaScript">
function mianimacion(objest)
{
var indAct, x, y, xfin, yfin, blqAct;
var incX, incY;
for(indAct = 0;indAct < objest.puntos.length; indAct++)
      {
      blqAct = objest.puntos[indAct];
      x = blqAct.xAct;
      y = blqAct.yAct;
      xfin = objest.referX+indAct*10;
      yfin = objest.referY;
      incX = 6*objest.puntos.length/(1+indAct);
      incY = 6*objest.puntos.length/(1+indAct);
      if (x < xfin)
            {
            x += incX;
            if (x > xfin) x = xfin;
            }
      else{
            if (x > xfin)
                  {
                  x -= incX;
                  if (x < xfin) x = xfin;
                  }
            }
      if (y < yfin)
            {
```

```
            y += incY;
            if (y > yfin) y = yfin;
            }
      else {
            if (y != yfin)
                  {
                  y -= incY;
                  if (y < yfin) y = yfin;
                  }
            }
      blqAct.xAct = x;
      blqAct.yAct = y;
      blqAct.style.left = x;
      blqAct.style.top = y;
      blqAct.fin =(x == xfin) && (y == yfin)
      }
}
//El evento temporizador sólo entiende de variables globales
//por eso se declara aquí la variable estela
var estela;
function iniciarEstela(texto)
{
var puntos = new Array();
for (i=0; i< texto.length; i++)
      {
      document.write('<div id="l'+i+'" style="position:absolute; ');
      document.write('width:1px; height:1px; color: red; ');
      document.write('z-index:14; left: 0px; top:
0px">'+texto.charAt(i));
 document.write('</div>')
 puntos[i]="l"+i;
      }
estela = new oEstela( puntos, -1,10, -1, 20);
for (i=0; i< estela.puntos.length; i++)
      {
      estela.puntos[i].xAct = estela.referX;
      estela.puntos[i].yAct = estela.referY;
      estela.puntos[i].fin = false;
      }
document.onmousemove = new Function("estela.seguir()");
estela.tempor = setInterval('mianimacion(estela)',5);
}
```

b) Código HTML

En la página sólo debes colocar la llamada a la función que crea la estela y dispara la animación del texto.

```
<html>
<head>
<title>
Animando el título de tu página.  </title>
<!-- aquí va el código javascrip -->
</head>
<body color ="#FFFFFF" >
<script language="Javascript">
iniciarEstela("Felicidades");
</script>
<!--Contenido de tu página-->
</body>
</html>
```

R) Buscador en la Página

Este sencillo script te permitirá colocar un buscador para que tus visitantes puedan encontrar un texto cualquiera en tus páginas. Funciona de manera muy similar a CTRL+F o la opción **Buscar en esta página** del menú **Edición** de Internet Explorer.

a) Código JavaScript

Cuando un visitante desea buscar un texto determinado en nuestra página puede usar como sabes una de las opciones del menú Edición del Navegador. Pero ¿no sería interesante ofrecerle un pequeño formulario para realizar esa búsqueda?. Pues este pequeño script te permite hacerlo. Se trata de aprovechar el objeto **textRange**, exclusivo de Microsoft Internet Explorer. Este objeto se usa para manipular una porción de texto incluida en el documento HTML, texto que puede ser toda la sección **body**. El funcionamiento de la función **buscarTxt** es muy simple: crea un objeto **textRange** con el elemento **body** de la página guardándolo en contenido, lo copia en el objeto textRange **temporal**, luego usa el método **findText** para encontrar el texto pasado como argumento. Una vez hallado pregunta si se desea continuar la búsqueda, en caso afirmativo modifica **contenido** para que englobe al texto en el que aún no se ha buscado y repite el proceso hasta que se le diga que pare o no encuentre más coincidencias.

Una observación en cuanto a los argumentos, ambos opcionales, del método: el primer argumento indica el número de caracteres a buscar y si es negativo indica buscar hacia atrás, y el segundo argumento indica como ha de ser la búsqueda:
0: Encuentra coincidencias parciales
2: Busca palabras completas
4: Mayúsculas y minúsculas no se consideran iguales.

```
function buscarTxt(texto)
{ if(!document.all) return
var contenido = document.body.createTextRange();
var seguir = true;
var temporal = contenido.duplicate();
var existe = temporal.findText(texto,0,0)
while(existe && seguir)
{
temporal.scrollIntoView(true)
temporal.select();
//Modifica contenido para que comience al final
//de la palabra encontrada
contenido.setEndPoint("StartToEnd", temporal);
seguir = confirm("¿Continuar la búsqueda?")
if (seguir)
{
//Repite labúsqueda
temporal = contenido.duplicate();
existe = temporal.findText(texto,0,0)
}
}
if (seguir)
alert("Fin del documento")
}
```

b) Código HTML

En esta página de ejemplo sólo se ha colocado un formulario con un campo de texto y un botón para comenzar la búsqueda. El evento **onclick** en este botón tiene vinculada la función **buscarTxt**, por lo que al pulsarlo llamará a esta función usando como argumento el valor contenido en el campo de texto. Lógicamente la página deberá contener texto donde realizar la búsqueda.

```
<html>
<head>
<title>Buscando </title>
<script language="Javascript">
<!-- aquí va el código javascript -->
</script>
</head>
<body color ="#FFFFFF" >
<form name="form1" method="post" action="">
<input type="text" name="texto">
<input type="button" name="Button" value="Buscar"
onclick="buscarTxt(this.form.texto.value)">
</form>
<!-- Resto de la página -->
</body></html>
```

S) Página de Inicio

Este pequeño script te permite colocar un botón o un enlace en tu página para sugerir al visitante que coloque tu sitio como página de inicio en su navegador.

a) Código JavaScript

Mediante este sencillo código puedes tener en tu página un enlace para que los usuarios coloquen tu página como página de inicio en su navegador. Hace uso de los **behaviors**, una característica exclusiva del explorador de Microsoft presente a partir de la versión 5. El código comprueba en primer lugar si el navegador es MSIE 5+ y soporta los behaviors, si es así simplemente escribe en la página el enlace con la frase aclaratoria de lo que hace el mismo. En lugar de texto puedes colocar un botón o una imagen, a fin de cuentas no es más que un enlace que actualiza el valor de **style.behavior** y llama al método **setHomePage** con la dirección URL que queremos colocar como página de inicio por defecto del navegador.

```
<script language="JavaScript">
var verBase = navigator.appVersion;
var url = "'http://www.misitio.com/'";
verBase = verBase.match(/MSIE ([^;]+);/);
verBase =parseFloat(verBase[1]);
if(navigator.appVersion.indexOf('MSIE') >-1 && verBase >=5)
{
with(document){
      write('<A HREF="#" ');
      write('onClick="this.style.behavior=');
      write("'url(#default#homepage)';")
      write('this.setHomePage('+url+');">');
      write('Pon mi sitio en tu página de inicio</a>')
      }
}
</script>
```

b) Código HTML

Como ves este script está situado directamente en el cuerpo del documento, puedes ponerlo en el lugar que desees. También podrías haber usado una función para el código y colocar una llamada en la sección body. Cuestión de gustos. Lo que no debes olvidar es que el script produce texto y debe ejecutarse a medida que se carga la página. Cuando el usuario pulsa el enlace le aparecerá un cuadro de diálogo para que confirme la elección, esto es inevitable por cuestiones de seguridad.

```
<html>
<title>Título de tu página</titulo>
<head>
</head>
<body>
<!--Coloca aquí el código y no olvides
las etiquetas <script>-->
</body>
</html>
```

T) Carrusel de Imágenes

Con este script podrás colocar en tu página un carrusel de imágenes con el que podrás mostrar de forma secuencial toda una lista de imágenes. Ideal para galerías, thumbplus, etc.

a) Código JavaScript

Esta colección de funciones nos permitirá colocar en nuestra página un proyector de imágenes con el que el visitante puede ir viendo una por una todas las imágenes de una serie. El sistema permite que sea el visitante el que vaya avanzando o retrocediendo por la colección de imágenes o bien puede usar un modo automático con el que las imágenes van cambiando cada cierto tiempo. Muy útil para galerías de fotos, por ejemplo.

El script lo que hace, es recorrer de forma controlada una lista con las direcciones de las imágenes que queramos mostrar, **lista** que se implementa como un **Array**. Para recorrerla se usa la función **cambio(sen)**, en la que el argumento indica si vamos hacia adelante o hacia atrás. Esta función se ejecuta al pulsar sobre los correspondientes links que se colocan en la página.

La otra función importante es la del cambio automático de imágenes, la función **automat(miliseg)**. Esta tan sólo inicia un temporizador que llama a la función **pasar()** con el intervalo de tiempo indicado en el argumento. La función **parar()** se usa para detener el cambio automático de las imágenes deteniendo el **temporizador**.
La función **reset()** tan sólo reinicia las variables globales y coloca la imagen inicial. Las variables globales usadas son la **lista** de imágenes, un **contador** para recorrer la lista, la variable **tiempo** para almacenar el intervalo para el cambio automático de las imágenes y **tempor** para almacenar el temporizador.

```
<script language="JavaScript">
<!--
var lista = new Array('1.jpg', '2.jpg', '3.jpg' , '4.jpg');
var contador = 0;
var tiempo = 0;
var tempor = null;
function cambio(sen)
{
contador+= sen;
if (contador ==lista.length) contador = lista.length-1;
else
if (contador < 0) contador = 0;
document.images.centro.src = lista[contador]
window.status="Imagen número "+contador
}
function reset()
{
contador = 0;
fin = false;
document.images.centro.src = lista[0]
}
function automat(seg)
{
tiempo = parseInt(seg);
reset();
window.status="Pase automático";
if (isNaN(tiempo) || tiempo <= 0)
alert("Error en el tiempo")
else
tempor = setTimeout("pasar()", tiempo)
}
function parar()
{
clearTimeout(tempor);
}
function pasar()
{
cambio(1);
if (contador < lista.length-1)
tempor = setTimeout("pasar()", tiempo)
}
// -->
</script>
```

b) Código HTML

Para ilustrar este script esta página usa una tabla con dos filas y tres columnas. En la columna central de la fila superior se encuentra la imagen que irá cambiando, y las celdas laterales contienen los enlaces que llaman a la función **cambiar()**. Estos enlaces puede cambiarlos por iconos para mejorar la estética. En la fila inferior, se encuentra el formulario que permitirá al visitante ver el paso de imágenes de forma automática, para ello deberá teclear un número de milisegundos y pulsar el botón inicio, desde ese momento las imágenes irán cambiando solas.

```
<html>
<head>
<title>Animando el título de tu página.  </title>
<!-- aquí va el código javascript -->
```

```
</head>
<body color ="#FFFFFF">
<table width="82%" border="0" align="center">
<tr>
<td width="30%" align="right" height="247"><font color="#0033CC"><b>
<a href="javascript:cambio(-1)">atras</a>
</b></font></td>
<td align="center" width="48%" height="247"><img id="centro"
src="1.jpg" height="170"></td>
<td width="22%" height="247"><font color="#0033CC"><b>
<a href="javascript:cambio(1)">avance</a>
</b></font></td></tr>
<tr>
<td width="30%" align="right"><font color="#0033CC"></font></td>
<td width="48%" align="center"><b><font
color="#0033CC">Automático</font></b>
<form name="form1" method="post"
action="javascritp:automat(this.form1.tiempo.value)">
<input type="text" name="tiempo">
<br>
<input type="submit" name="Button" value="Inicio">
<input type="button" name="Button2" value="Parar" onclick="parar()">
</form>
</td>
<td width="22%"><font color="#0033CC"></font></td>
</tr>
</table>
```