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.

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 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);
```

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);
```

OBJETO MATH

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

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 de la 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

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
50
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).

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:

ToExponential().	Convierte el número en una notación exponencial.
toFixed().	Formatea el número con la cantidad de dígitos decimales que pasemos como parámetro.
ToPrecision().	Formatea el número con la longitud que pasemos como parámetros.

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,

OBJETO STRING

Propiedades

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

Métodos:

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".

OBJETO NAVIGATOR

Este objeto proporciona información sobre el navegador que está utilizando el usuario (nombre, versión, sistema operativo que lo soporta, etc.).

Este objeto es importante debido a que dependiendo de las versiones del navegador (IExplorer, Netscape, Opera, Mozilla, etc) se soportan diferentes comandos y sintaxis de JavaScript. De esta forma es posible personalizar la ejecución del código en función de los distintos navegadores.

Propiedades:

appCodeName.	Cadena que contiene el nombre del código interno del navegador. Por razones que no vienen al caso suele ser mozilla.
appName.	Cadena que contiene el nombre oficial del Navegador del cliente.
appVersion.	Cadena que contiene información sobre la versión del Navegador del cliente. Hay que utilizar funciones de cadena para extraer la información importante.
cookieEnabled	Propiedad booleana que devuelve true sei el navegador está configurado para permitir cookies, y false si no lo está.
platform.	Cadena con la plataforma sobre la que se está ejecutando el programa cliente Win 16(Windows 3.x), Win 32((Windows 9x, Me, NT, XP y 2000), MacXXX (Para S.O Macintosh).
userAgent	Cadena que incluye otras propiedades como: appcodeName, appName, appVersion, lenguaje y platform.
Importante: Esta es la cadena que se debe examinar para extraer información correcta acerca del navegador.	Está cadena se corresponde con la cadena de encabezamiento USER_AGENT del protocolo HTTP que transmite el navegador cuando solicita petición al servidor.
	Por razones históricas, el agente de usuario comienza siempre con la palabra Mozilla, aunque el navegador no tenga nada que ver con esta organización.

Métodos:

Métodos de Navigator: javaEnable().

Informa si el navegador está habilitado para soportar la ejecución de programas escritos en Java.

OBJETO SCREEN

Este objeto almacena información acerca de la configuración de pantalla del usuario.

Es de una gran utilidad, sobre todo a la hora de ver que configuración, respecto a la resolución tiene el usuario, y de esa manera poder amoldar nuestra página a dicha configuración.

Todas las propiedades de este objeto son de modo lectura, es decir, que no podremos asignarlas un valor.

height: .	Devuelve la altura en píxel que tiene la configuración de nuestra pantalla
width:	Devuelve el ancho en píxel que tiene la configuración de nuestra pantalla.
availHeight:	Devuelve la altura disponible para el diseño. Si hay una aplicación (como la barra de Windows) de modo visible, nuestra altura de diseño no es la de toda la pantalla (height), sino que es la de toda la pantalla, menos la altura de la barra de estado.
availWidth:	Devuelve el ancho disponible para el diseño. Sucede lo mismo que con la propiedad availHeight.
colorDepth:	Indica los bits de profundidad (2, …16,24,32) utilizada.

OBJETO WINDOW

Propiedades del objeto Window:

closed	Es un booleano que nos dice si la ventana está cerrada (closed = true) o no (closed = false)
defaultStatus	Cadena que contiene el texto por defecto que aparece en la barra de estado (status bar) del navegador.
document	Objeto que contiene el la página web que se está mostrando
frame	Marco de la página web, se accede pr su nombre
history	Se trata de un array que representa las URLS almacenadas en su historial
innerHeight	Tamaño en pixels del espacio donde se visualiza la página, en vertical. También Height.
innerWidth	Tamaño en pixels del espacio donde se visualiza la página, en horizontal. También Width.
length	Variable que nos indica cuántos frames tiene la ventana actual.
location	Cadena con la URL de la barra de dirección. Podemos cambiar el valor de esta propiedad para movernos a otra página. Ver también la propiedad location del objeto document.
locationbar	Objeto barra de direcciones de la ventana.
menubar .	Objeto barra de menús de la ventana
name	Contiene el nombre de la ventana, o del frame actual.
opener	Es una referencia al objeto window que lo abrió, si la ventana fue abierta con el método open().
outherHeight .	Tamaño en pixels del espacio de toda la ventana, en vertical. Esto incluye las barras de desplazamiento, botones, etc
outherWidth	Tamaño en pixels del espacio de toda la ventana, en horizontal.

	Esto incluye las barras de desplazamiento.
parent	Referencia al objeto window que contiene el frameset. Hace referencia a la ventana donde está situada el frame donde estamos trabajando.
personalbar	Objeto barra personal del navegador. (Javascript 1.2)
self	Es un nombre alternativo del window actual.
status	String con el mensaje que tiene la barra de estado
pageXOffset	Total de pixels que está desplazado horizontalmente el documento
pageYOffset	Total de pixels que está desplazado verticalmente el documento
scrollbars	Objeto de las barras de desplazamiento de la ventana.
top	Nombre alternativo de la ventana del nivel superior. Como parent
toolbar	Objeto barra de herramientas.

Métodos del objeto window:

back()	Emula el botón de 'página atrás' de la barra de herramientas del navegador
forward()	Emula el botón de 'página adelante' de la barra de herramientas del navegador
home()	Simula pinchar en el botón de página de inicio del navegador
stop()	Simula pinchar el botón de stop de la ventana del navegador
alert(mensaje)	Muestra un mensaje en un cuadro de diálogo que contiene un botón Aceptar
blur()	Elimina el foco del objeto window actual.
clearInterval(id)	Elimina el intervalo referenciado por 'id' (ver el método setInterval()).
clearTimeout(nombre)	Cancela el intervalo referenciado por 'nombre' (ver el método setTimeout()).

close()	Cierra el objeto window actual.
confirm(mensaje)	Muestra un mensaje en un cuadro de diálogo, contiene dos botones Aceptar y Cancelar que devuelven valores boolenanos.
focus()	Captura el foco del ratón sobre el objeto window actual.
moveBy(x,y)	Mueve el objeto window actual el número de pixels especificados de forma relativa por (x,y)
moveTo(x,y)	Mueve el objeto window actual a las coordenadas absolutas (x,y).
open(URL,nombre,caracteristicas)	Abre la URL que le pasemos como primer parámetro en una ventana de nombre nombre. Si la URL no existe, abrirá una ventana nueva. Las características para la ventana que queramos abrir son: • fullscreen=[yes|no] . Maximizar ventana • toolbar = [yes|no|1|0] .Si la ventana tendrá barra de herramientas (yes,1) o no (no,0). • location = [yes|no|1|0] .Si la ventana tendrá campo de localización o no. • directories = [yes|no|1|0] .Si la nueva ventana tendrá botones de dirección o no. • status = [yes|no|1|0] .Si la nueva ventana tendrá barra de estado o no. • menubar = [yes|no|1|0] Si la nueva ventana tendrá barra de menús o no. • scrollbars = [yes|no|1|0] Si la nueva ventana tendrá barras de desplazamiento o no. • resizable = [yes|no|1|0] Si la nueva ventana podrá ser cambiada de tamaño o no. • width = px El ancho de la ventana en pixels. • height = px Alto de la ventana en pixels. • outerWidth = px El ancho *total* de la ventana en pixels. • outerHeight = px

	El alto *total* de la ventana el pixels. • left = px La distancia en pixels desde el lado izquierdo de la pantalla a la que se debe colocar la ventana. • top = px La distancia en pixels desde el lado superior de la pantalla a la que se debe colocar la ventana. Para crear una ventana se utiliza la sintaxis siguiente: **var nombre_ventana;** **nombre_ventana=window.open([parametros]);**
prompt(mensaje,respuesta_por_defecto)	Muestra un cuadro de diálogo que contiene una caja de texto en la que el usuario puede introducir datos como respuesta al mensaje. El parámetro 'respuesta_por_defecto' es opcional, y mostrará la caja de texto con la respuesta por defecto indicada. Devuelve el valor introducido o cadena vacía
resizeBy(ancho,alto)	Redimensiona la ventana de forma relativa con los valores de alto y ancho con valores negativos o positivos.
resizeTo(ancho,alto)	Redimensiona la ventana de forma absoluta a los valores, ancho alto. No hay que tener la ventana maximizada.
scrollBy(x,y)	Desliza el documento dentro de la ventana con incrementos (x,y) negativos o positivos.
scrollTo(x,y)	Desliza el documento dentro de la ventana para dejarlo en las coordenadas especificadas por (x,y).
setInterval(expresion,tiempo)	Evalúa la expresión especificada después de que hayan pasado el número de milisegundos especificados en tiempo. Devuelve un valor que puede ser usado como identificativo por clearInterval().
setTimeout(expresion,tiempo)	Evalúa la expresión especificada después de que hayan pasado el número de milisegundos especificados en tiempo. Devuelve un valor que puede ser usado como identificativo por clearTimeout().
print()	Imprime el documento

OBJETO DOCUMENT

Con el objeto document se controla la página web y todos los elementos que contiene, representa a la página actual que se está visualizando en ese momento.

Depende del objeto window, o del objeto frame en caso de que la página se esté mostrando en un frame.

alinkColor	Devuelve o asigna el color de los enlaces activos
anchors	Devuelve un array de las anclas del documento.
applets .	Devuelve un array con todos los applets de la página
bgColor	Devuelve o asigna el color de fondo del documento.
cookie	Devuelve o asigna una cookie
domain	Devuelve o asigna el nombre del dominio del servidor de la página.
embeds	Devuelve todos los elementos de la página incrustados con <EMBED>. (Javascript 1.1)
fgColor	Devuelve o asigna el color del texto. Para ver los cambios hay que reescribir la página.
forms .	Devuelve o asigna un array con todos los formularios de la página
images	Devuelve o asigna las imágenes de la página introducidas en un array.
lastModified .	Devuelve a fecha de última modificación del documento
layers	Corresponde a las capas etiqueta <layer> del documento.
linkColor	Devuelve o asigna el color de los enlaces.
links	Devuelve un array con cada uno de los enlaces de la página.
plugins	Corresponde a las referencias y llamadas de los plugins del documento.
referrer	Devuelve la página de la que viene el usuario. Si es la primera devuelve null.
title	El titulo de la página.
URL	Devuelve o asigna la direccion del documento, similar a Location.
vlinkColor	El color de los enlaces visitados.

Métodos del objeto Document

captureEvents()	Intercepta un evento para que pueda ser interceptado por el documento
open() .	Abre el flujo del documento
close() .	Cierra el flujo del documento
write() .	Escribe dentro de la página web. Podemos escribir etiquetas HTML y texto normal
writeln()	Escribe igual que el método write(), aunque coloca un salto de línea al final.
getselection()	Devuelve el texto seleccionado en el documento.
handleEvent()	Activa el manejador del evento especificado
home()	Carga la página de inicio
releaseevents()	Libera los eventos que han sido interceptados
routeEvents()	Intercepta un evento y lo pasa a lo largo de la jerarquía del objeto que lo lanzó.

.

OBJETO HISTORY

Contiene un array que almacena las URLs que el usuario ha visitado en la ventana actual. Mediante los métodos que posee este objeto se puede navegar adelante o atrás en el historial.

Propiedades

current	Especifica la URL actual del historial.
length	Indica el número de entradas que almacena el historial.
next	Especifica la URL de la siguiente entrada en el historial.
previous	Especifica la URL de la entrada previa en el historial.

.

Métodos

back()	Carga la URL previa del historial.
forward()	Carga la siguiente URL del historial.
go()	Carga la URL indicada en el historial.

OBJETO LOCATION

Representa la URL completa asociada a un objeto window. Cada una de las propiedades de este objeto representa una porción de la URL. Se accede mediante la propiedad location de un objeto window.

Una URL se compone de las siguientes partes: protocol://host:port/pathname#hash?search

Propiedades del objeto location:

hash	Especifica el valor de un ancla en una URL.
host	Especifica el nombre de dominio o dirección IP de la URL.
hostname	Especifica la parte host:port de la URL.
href	Especifica la URL entera.
pathname	Especifica la ruta de la URL.
port	Especifica el puerto de comunicaciones que usa el servidor.
protocol	Especifica el protocolo de la URL.
search	Especifica la consulta incluida en la URL

.
Métodos del objeto location:

reload()	Recarga el documento actual de la ventana.
replace()	Carga la URL especificada sobre la entrada actual del historial de navegación.
assing()	Carga un nuevo documento