Curso de JavaScript

Lola Cárdenas Luque
http://rinconprog.metropoli2000.com
lornacl@iname.com
Última actualización: 3 de febrero de 2001

Índice General

1 Introducción

Todos los que hasta ahora hayan seguido el curso de HTML, se habrán dado cuenta de una cosa: crear un documento HTML es crear algo de caracter estático, inmutable con el paso del tiempo. La página se carga, y ahí termina la historia. Tenemos ante nosotros la información que buscábamos (o no ;)), pero no podemos **INTERACTUAR** con ella.

Surge después la interfaz CGI que, unida a los formularios, comienza a permitir un poco de interactividad entre el cliente (quien está navegando) y el servidor (quien aloja las páginas). Podemos rellenar un formulario y enviárselo al servidor, teniendo de esta manera una vía de comunicación.

Sin embargo, para hacer esto (enviar un formulario) necesitamos hacer una nueva petición al servidor quien, cuando la procese, nos enviará (si procede) el resultado. ¿Y si nos hemos olvidado de rellenar algún campo? Cuando el servidor procese la información, se dará cuenta de que nos hemos olvidado de rellenar algún campo importante, y nos enviará una página con un mensaje diciendo que nos faltan campos por rellenar. Tendremos que volver a cargar la página, rellenar el formulario, enviarlo, el servidor analizarlo, y, si esta vez no ha fallado nada, nos dará su respuesta.

Todo esto supone ***recargar innecesariamente la red*** si de alguna manera desde el propio cliente existiera una forma de poder comprobar esto **antes** de enviar nuestra petición al servidor, con el consiguiente ahorro de tiempo.

Buscando la interactividad con el servidor, surgen lenguajes destinados a ser usados en la red. Uno de ellos es el conocido lenguaje Java, o la tecnología ActiveX. Sin embargo, ambos tienen el mismo problema: para alguien no iniciado, el aprendizaje de alguna de estas opciones supone un esfuerzo considerable. Además, el volumen de información que debe circular por la red al usar este método, vuelve a hacer que los tiempos de carga resulten largos y por tanto poco adecuados (hemos de recordar que el teléfono NO es gratis, ni tan siquiera parecido).

Así pues, como solución intermedia, nace JavaScript. ¿Y qué es JavaScript?

Se trata de un lenguaje de tipo script compacto, ***basado en objetos y guiado por eventos*** diseñado específicamente para el desarrollo de aplicaciones cliente-servidor dentro del ámbito de Internet.

Los programas JavaScript van **incrustados** en los documentos HMTL, y se encargan de realizar acciones en el cliente, como pueden ser pedir datos, confirmaciones, mostrar mensajes, crear animaciones, comprobar campos...

Al ser un lenguaje de tipo script significa que **no es** un lenguaje **compilado**, es decir, tal cual se va leyendo se ejecuta por el cliente. Al estar basado en objetos, habrá que comentar (en otra entrega) qué son los objetos, aunque no vamos a tener toda la potencia que estos nos dan en Java, sólo algunas de sus características. Estar guiado por eventos significa que no vamos a tener un programa que se ejecute de principio a fin en cuanto carguemos

la página web. Significa que, cuando en el navegador suceda algún evento, entonces, si lo hemos decidido así, pasará ALGO. Y ese algo será alguna función JavaScript. Al ser guiado por eventos, no tenemos una función principal que se ejecute por delante de las demás, sino que tendremos funciones, y, por ejemplo, si pulso el ratón sobre un cierto enlace, entonces se ejecutará una función, pero si pulso sobre una zona de una imagen sensible puede ejecutarse otra función.

El programa que va a interpretar los programas JavaScript es el propio navegador, lo que significa que si el nuestro no soporta JavaScript, no podremos ejecutar las funciones que programemos. Desde luego, Netscape y Explorer lo soportan, el primero desde la versión 2 y el segundo desde la versión 3 (aunque las primeras versiones de Explorer 3 soportaban una versión propia del lenguaje llamada JScript y con la que, para qué dudarlo, había incompatibilidades con el verdadero JavaScript).

Sin embargo, dado que es un lenguaje para usar en la red, y hoy por hoy las dos grandes compañías se la disputan con sus navegadores, está muy sometido a las presiones y pequeñas incorporaciones que cada una de ellas quiera añadirle. Esto es así porque no tenemos un estándar de JavaScript perfectamente definido (o teníamos, que ya estoy un poco anticuada y no sé seguro si ya hemos llegado al estándar o no, aunque yo creo que no). De todas formas, no hay por qué preocuparse, ya que lo que se va a comentar en este curso *debería* funcionar en cualquier navegador que soporte JavaScript (lo probaré primero antes de decir "esto funciona" y que luego no funcione, como casi todos los ejemplos que podeis encontrar en libros del tema que no sean serios).

Ahora que ya conocemos un poco qué es lo que vamos a utilizar, y sus batallitas, ya podemos, en la siguiente entrega, empezar a ver cómo programar en JavaScript.

PD: Mirad algo de C en algún curso que tengais, porque las semejanzas no se explicarán con detalle (así podemos centrarnos en la materia). De todas formas, si salen dudas, siempre podeis preguntarme.

2 Tipos de datos. Estructuras de control. Declaración de funciones

Lo primero que uno debe saber antes de programar nada en JavaScript, es cómo incrustarlo en un documento HTML. Para ello, tenemos dos formas; una, usando una directiva especial y otra, como parámetro de ciertas directivas (como <A>, <IMG>,...) en respuesta a eventos de usuario ya predefinidos (pulsar el ratón, cargarse la página,...).

El primer método consiste en usar la directiva pareada <SCRIPT>. Esta directiva admite como parámetro LANGUAGE, que en principio sólo sería JavaScript, pero como ahora también existe el VBScript hay que tenerlo en cuenta. Además, no hemos de olvidar que habrá quien tenga un navegador que no soporte JavaScript y habrá quien sí pero no quiera soportarlo, así que, de alguna manera tenemos que prevenir esto, evitando que todo el código salga desparramado en la pantalla del navegador.

Bien, pues para ello encerraremos el código JavaScript entre comentarios, pero haciéndolo de esta forma para no tener problemas:

```
<SCRIPT LANGUAGE="JavaScript">
 <!--
    Codigo JavaScript
 //-->
</SCRIPT>
```

¿Y dónde colocamos este código?

En principio no importa mucho dónde lo pongamos, sin embargo, una **buena costumbre** es introducirlo en la **cabecera** del documento HTML (ya sabeis, entre <HEAD> ... </HEAD>), puesto que así estais completamente seguros de que cuando se empieza a cargar el documento y a aparecer en la pantalla, todo el código JavaScript ya está cargado y listo para ser ejecutado si se da el evento apropiado.

La otra forma de introducir código JavaScript en nuestros documentos HTML es en respuesta a determinados eventos que puede recibir el documento como consecuencia de las acciones del usuario que esté viendo la página en ese momento. Estos eventos están asociados a ciertas directivas HTML. La forma general será esta:

```
<DIRECTIVA nombreEvento="Codigo_JavaScript">
```

Vamos a ver un ejemplito antes de que salgais corriendo al ver esta cosa tan rara O:)

Hay una función JavaScript de la que ya hablaremos, cuyo nombre es `alert()`. Esta función muestra una ventanita por pantalla con el mensaje que nosotros le pasemos como argumento.

Un posible evento es que el ratón pase sobre una cierta zona del documento HTML (`onMouseOver`). Parece obvio pensar que este evento estará asociado a los enlaces (por ejemplo).

Bien, pues como ejemplo, veamos cómo hacer para que cuando el ratón se situe sobre un link nos salga una ventanita con un mensaje:

```
<A HREF="doc.html" onMouseOver="alert('Por aqui no pasas');">Pasa
    por aqui si eres valiente</A>
```

cuyo resultado es: Pasa por aquí si eres valiente

Un consejo: no useis esto en vuestras páginas web. Es un tocamorales de mucho cuidado, sólo lo he puesto como ejemplo con fines didácticos, pero es altamente desaconsejable que lo useis. Avisados quedais :)

Ya trataremos en su momento los eventos. Por ahora, vamos a hacer un resumen de la sintaxis del lenguaje. Como ya avisé, es muy parecida a la del C, por lo que no voy a explicar los aspectos en los que coincida (ya lo comenté en el capítulo anterior).

Antes de comenzar a hablar del lenguaje, tengo que dar unas pequeñas notas (el maravilloso mundo de los navegadores `;-DDD`)

Como hemos visto, en la directiva <SCRIPT> hemos especificado un parámetro, `LANGUAGE`, que tomaba el valor `''JavaScript''`. Si escribimos esto, será interpretado como que estamos escribiendo código de la versión 1.0 para el Navegador Netscape 2.0 en adelante, para el Netscape 3.0 en adelante estamos usando la versión 1.1 de JavaScript, y para Netscape 4.0 en adelante interpreta que estamos usando la versión 1.2 del lenguaje.

Resumiendo: a no ser que utilicemos cosas de versiones avanzadas del lenguaje, basta con que pongamos `'<SCRIPT LANGUAGE=''JavaScript''>'`, pero si queremos asegurarnos de que un navegador antiguo no tenga problemas con el código de una versión nueva, escribiremos `'<SCRIPT LANGUAGE=''JavaScript 1.x''>'`, para que este navegador, al no reconocer el lenguaje, ignore el script y nos ahorre problemas.

2.1 Sintaxis de JavaScript

El lenguaje presenta los siguientes elementos:

- Es "Case Sensitive", es decir, distingue mayúsculas de minúsculas.
- Comentarios: igual que en C/C++, `/* ... */` para encerrar un bloque y `//` para comentarios de una línea.
- Cada sentencia ha de terminar en ;
- Encerrando código entre llaves { ... } lo agrupamos.

2.2 Variables

JavaScript tiene la peculiaridad de ser un lenguaje ***débilmente tipado***, esto es, uno puede declarar una variable que ahora sea un entero y más adelante una cadena.

Es decir, podremos hacer cosas como:

```
MiVariable=4;
```

y después:

```
MiVariable="Una_Cadena";
```

Para declarar variables no tenemos más que poner la palabra var y a continuación la lista de variables separadas por comas. No todos los nombres de variable son válidos, hay unas pocas restricciones:

- Un nombre válido de variable no puede tener espacios.
- Puede estar formada por números, letras y el caracter subrayado _
- No se puede usar palabras reservadas (`if`, `for`, `while`, `break`...).
- No pueden empezar por un número, es decir, el primer caracter del nombre de la variable ha de ser una letra o _

Ejemplos de **definiciones erróneas**:

```
var Mi Variable, 123Probando, $Variable, for, while;
```

Ejemplos de definiciones **correctas**:

```
var _Una_Variable, P123robando, _123, mi_carrooo;
```

Por supuesto, podemos inicializar una variable al declararla, como se hace en C:

```
var Una_Variable="Esta Cadenita de texto";
```

Visto lo que debemos saber sobre las variables, veamos ahora de qué tipos de datos disponemos en JavaScript.

2.3 Tipos de Datos

Cadenas: En JavaScript, las cadenas vienen delimitadas o bien por comillas dobles, o bien por comillas simples, y pueden tener cualquier combinación de letras, espacios, números y otros símbolos.

¿A qué se debe que sea válido tanto encerrar las cadenas entre comillas dobles como encerrarlas entre comillas simples?

Muchas veces, habrá alguna función JS incrustada en alguna directiva HTML, ***y esta usa las comillas dobles;*** en este caso, **usaremos las comillas simples** (como en el ejemplo del alert visto más arriba) **para no cerrar el valor del parámetro que estábamos dando a la directiva HTML.**

Podemos, además, usar los caracteres de escape que tenemos en C para representar los saltos de línea, tabulaciones, etc...

Los más usados serán estos:

```
\b    \f    \n    \r    \t    \\    \'    \"
```

Enteros

Coma flotante

Hexadecimales y octales : Para denotar un número hexadecimal lo haremos escribiendo delante del número **0x** (por ejemplo, `0xF3A2` se refiere a `F3A2` en hexadecimal). Para denotar un número en base octal lo haremos precediéndolo de un **0** (por ejemplo, `01763` se refiere a `1763` en octal).

Booleanos: Tomarán los valores **true** o **false** (y no 1 ó 0).

Nulos: El valor **null** nos puede servir para averiguar si hemos inicializado o no una variable, comprobándolo con un `if (cf. Section 2.5.1)`.

2.4 Operadores

Vamos a ver, por grupos, de qué operadores disponemos en JS para poder hacer operaciones con nuestras variables.

2.4.1 Operadores Aritméticos

Suma (+), resta (-), multiplicación (*), división (/) y módulo (%)

Llevad cuidado con el módulo puesto que con números negativos no trabaja como debiera; con positivos no hay problema.

Cuando estamos utilizando cadenas, el operador `+` tiene el significado "**concatenación**". Por ejemplo, si escribimos

```
var cadena="Suma"+"_de_"+"cadenas";
```

es lo mismo que si hubiéramos escrito

```
var cadena="Suma_de_cadenas";
```

2.4.2 Incrementos y decrementos

Son los conocidos $++$ y $--$

Notar que, si tenemos definidas dos variables, por ejemplo:

```
var Variable1, Variable2;
```

no es lo mismo hacer esto:

```
Variable2 = 20;
Variable1 = Variable2++;
```

que hacer esto:

```
Variable2 = 20;
Variable1 = ++Variable2;
```

en el primer caso, se realiza primero la asignación y después el incremento (con lo que tendremos que `Variable1=20`, `Variable2=21`); en el segundo caso se realiza primero el incremento y después la asignación, así que ambas valen lo mismo.

2.4.3 Otros operadores

Negación (-), AND lógico (&&), OR lógico (||), NOT lógico (!), AND bit a bit (&), OR bit a bit (|), XOR bit a bit (^), NOT bit a bit (~), rotación a derecha bit a bit (>>),

rotación a izquierda bit a bit (<<), igual (==), distinto (!=), mayor que (>), menor que (<), mayor o igual que (>=), menor o igual que (<=), coma (,).

Como veis, son los mismos que en el lenguaje C, así que yo no me voy a extender más en este punto. Con respecto a la precedencia, siguen el mismo orden de preferencia que tienen en el lenguaje C, pudiéndose alterar con el correcto uso de los paréntesis.

Tenemos también una funcioncita, **typeof(argumento)**, que nos devuelve una cadena que describe el tipo de dato que corresponde con el argumento `'argumento'` que se le pasa como parámetro. Esta función pertenece al lenguaje a partir de la versión 1.1.

Pasamos a otro punto importante, como son las estructuras de control del lenguaje.

2.5 Estructuras de control en JavaScript

2.5.1 Condicional:

```
if(condicion) {
 codigo necesario }
else {
 codigo alternativo }
```

La condición de comparación puede ser compuesta, y podemos tener varios `if` anidados.

También tenemos el condicional ternario:

```
[ expresion ] ? [ sentencia_1 ] : [ sentencia_2 ] ;
```

que evalúa [`expresión`]. El resultado es un booleano. Si es `'true'` se ejecutará [`sentencia_1`] y si es `'false'`, se ejecutará [`sentencia_2`]

Por ejemplo,

```
(x>y) ? alert("el mayor es x") : alert("el mayor es y");
```

si es cierto que `x>y`, entonces se ejecutaría la sentencia `'alert(''el mayor es x'')'`, pero si no es cierto, se ejecutaría `'alert(''el mayor es y'')'`.

2.5.2 Bucles:

```
for([inicializacion];[condicion];[expresion de actualizacion])
 {
```

```
instrucciones a repetir
}
```

```
while(condicion) {
 instrucciones a repetir
 }
```

El bucle `do` forma parte del lenguaje a partir de la versión 1.2

```
do {
 instrucciones a repetir
 } while(condicion);
```

Podemos romper los bucles con `'break'` y con `'continue'` de la misma forma que se rompen los bucles en C.

A partir de la versión 1.2, podemos usar etiquetas en el código (las etiquetas, igual que en C, tienen el formato `'Nombre_etiqueta:'`), y las sentencias `break` y `continue` pueden hacer referencia a una etiqueta (escribiendo `'break Nombre_etiqueta'` y `'continue Nombre:etiqueta'`).

También tenemos el switch (a partir de JS 1.2):

```
switch(condicion) {
  case caso_1 : sentencias para caso_1; break;
       ......
  case caso_N : sentencias para caso_N; break;
  default : sentencias por defecto; break;
  }
```

Además, disponemos de:

```
for(var 'propiedad' in 'objeto') {
  instrucciones a repetir
  }
```

que produce una iteración a través de todas las propiedades de un objeto (quien no sepa qué es un objeto que no desespere, en el siguiente capítulo se habla del tema), y también tenemos:

```
with(objeto) {
  varias sentencias
  }
```

que nos ahorra escribir siempre el nombre de un objeto si vamos a utilizarlo muchas veces para acceder a sus métodos, propiedades,... (mismo comentario sobre objetos que antes `O:)`).

Veremos la utilidad del `with` cuando usemos, por ejemplo, el objeto `Math` (calma, calma, todo a su tiempo `;)`).

2.6 Funciones

En JavaScript también podemos definir funciones (por medio de la palabra reservada function), pasarles argumentos y devolver valores. La estructura general de la definición de una función es como sigue:

```
function Nombre_Funcion(arg1, ..., argN)  {
 codigo de la funcion

 return valor;
 }
```

Podremos llamar a nuestra función (definida en la cabecera del documento HMTL) como respuesta a algún evento, por ejemplo:

```
<A HREF="doc.htm" onClick="Nombre_Funcion(arg1,...);">Pulse
por aqui, si es tan amable</A>
```

Las funciones en JavaScript tienen una propiedad particular, y es que **no tienen un número fijo de argumentos**. Es decir, nosotros podremos llamar a las funciones con un número cualquiera de argumentos y con cualquier tipo de argumentos.

Los argumentos pueden ser accedidos bien por su nombre, bien por un vector de argumentos que tiene asociada la función (que será **Nombre_Funcion.arguments**), y podemos saber cuántos argumentos se han entrado viendo el valor de **Nombre_Funcion.arguments.length**

Por ejemplo:

```
function Cuenta()  {
 alert("Me han llamado con " + Cuenta.arguments.length +
       " argumentos");
 for(i=0;i<Cuenta.arguments.length;i++)
   alert("Argumento "+i+": "+Cuenta.arguments[i]);
}
```

La podemos llamar con:

```
<SCRIPT LANGUAGE="JavaScript">
<!--
 Cuenta("Hola","a","todos");
//-->
</SCRIPT>
```

Es un ejemplo un poco simple, pero ilustra lo que quiero decir :)

2.6.1 Ambito de las variables

En JS también tenemos variables locales y variables globales. Las variables **locales** serán aquellas que se definan dentro de una función, mientras que las variables **globales** serán aquellas que se definan fuera de la función, y podrán ser consultadas y modificadas por cualquiera de las funciones que tengamos en el documento HTML. Un buen sitio para definir variables globales es en la cabecera, <HEAD> ... </HEAD>

En otra entrega, trataré el tema de los arrays con más calma, puesto que la forma de declararlos es un tanto especial y ya no coincide con lo que se hace en C. De todas formas, necesito hablar de objetos para poder explicar los arrays... O:-D

2.7 Ejercicios

Para poder hacer ejercicios sin llenar la pantalla de ventanitas 'alert', os voy a comentar una función de JS que os permite escribir texto en la ventana del navegador (aunque hablaré más ampliamente de ella cuando le llegue el turno). Se trata de:

```
document.write(cadena);
```

que escribe en el navegador la cadena 'cadena'. Cadenas válidas son cualquiera, desde ''Pepito Grillo'' hasta ''
 <HR>'', no sé si me pillais... y si no me pillais, probadlo... ;-)

1. Escribid una función que concatene dos cadenas, y que muestre tanto las cadenas iniciales por separado como la cadena final en la ventana del navegador.
2. Escribid las tablas de multiplicar en el navegador usando cuantos bucles necesiteis.
3. Escribid una función que muestre, aseadamente, en la ventana del navegador, los números pares del 1 al 100. Aseadamente significa que no deben ir todos en la misma línea.
4. Escribid una función que calcule y muestre la suma de los 1000 primeros números naturales.

5. Escribid una función que vaya calculando las sumas de los n primeros números naturales, con n variando de 1 a 500, y que escriba en la pantalla del navegador aquellas sumas que sean impares.

3 Funciones propias del lenguaje

En el capítulo anterior hablábamos de los elementos del lenguaje. Sin embargo, todavía no se ha trazado de forma clara cómo con un lenguaje de este tipo se puede interactuar con los elementos de una página web. Eso es lo que comenzamos a tratar en este capítulo.

3.1 Funciones propias del lenguaje

Las funciones que se van a describir aquí son funciones que no están ligadas a los objetos del navegador, sino que forman parte del lenguaje. Estas funciones nos van a permitir convertir cadenas a enteros o a reales, evaluar una expresión...

```
parseInt(cadena,base);
```

Esta función devuelve un entero como resultado de convertir la cadena que se le pasa como primer argumento. Opcionalmente, podemos indicarle la base en la que queremos que nos devuelva este entero (octal, hexadecimal, decimal, ...). Si no especificamos base, nos devuelve el resultado en base diez. Si la cadena no se puede convertir a un entero, la función devuelve el mayor entero que haya podido construir a partir de la cadena. Si el primer carácter de la cadena no es un número, la función devuelve `NaN` (Not a Number).

3.1.1 Ejemplos:

```
a = parseInt(''1234'');
```

En `'a'` tenemos almacenado el valor 1234, y podremos hacer cosas como `a++`, obteniendo 1235 ;)

Produce el mismo efecto que hacer

```
a = parseInt(''1234'',10);
```

puesto que, como hemos dicho, la base decimal es la base por defecto.

```
a = parseInt(''123cuatro5'');
```

En `'a'` tenemos almacenado el valor 123.

```
a = parseInt(''a1234'');
```

En `'a'` tenemos almacenado el valor `NaN`.

Vamos a jugar un poco con otras bases y ver los resultados:

```
a = parseInt(''FF'',16);
a = 255
```

```
a = parseInt(''EE'',15);
```

```
a = 224

   a = parseInt(''EE'',14);
a = NaN

   a = parseInt(''HG'',18);
a = 322
```

parseFloat(cadena);

Esta función nos devuelve un real como resultado de convertir la cadena que se le pasa como argumento. Si la cadena no se puede convertir a un real, la función devuelve el mayor real que haya podido construir a partir de ella. Si el primer carácter no puede ser convertido en un número, la función devuelve `NaN`.

3.1.2 Ejemplos:

```
a = parseFloat(''1.234'');
```
En 'a' tenemos almacenado el valor 1.234.

```
   a = parseFloat(''1.234e-1'');
```
En 'a' tenemos almacenado el valor 0.1234.

```
   a = parseFloat(''1.2e-1er45'');
```
En 'a' tenemos almacenado el valor 0.12.

```
   a = parseFloat(''e'');
a = parseFloat(''e-1'');
a = parseFloat(''.e-1'');
```

Todas ellas producen `a = NaN`.

```
   a = parseFloat(''.0e-1'');
a = 0

   a = parseFloat(''1.e-1'');
a = 0.1
```

escape(cadena);

Nos devuelve la secuencia de códigos de escape de cada uno de los caracteres de la cadena que le pasemos como argumento. Estos códigos serán devueltos con el formato `%NN`, donde `NN` es el código de escape en hexadecimal del caracter. Hay algunos caracteres que no dan lugar a este código de escape, sino que el resultado de escape sobre ellos vuelven a ser ellos. Algunos de estos son (obtenidos por inspección):

Los números (0 a 9). Las letras SIN acentuar (mayúsculas y minúsculas). Los siguientes, encerrados entre llaves (es decir, las llaves NO): { / . - _ * }

3.1.3 Ejemplos:

```
escape('$');
```
devuelve `%24`

```
escape(''a B c_1_%á'');
```
devuelve `a%20B%20c%20_1_%25%E1`

unescape(codigos);

Nos devuelve la cadena formada por los caracteres cuyo código le especificaremos. Los códigos deben ser especificados con el mismo formato con el que escape los proporciona: `%NN`, donde `NN` es el código en hexadecimal.

Por ejemplo:

```
unescape('a%0D%09b');
```
nos devuelve la cadena

```
a
        b
```

ya que `%0D` es el código del retorno de carro y `%09` el del tabulador.

También es válido hacer:

```
unescape('á$a');
```
y nos devuelve la misma cadena. Si metemos el `%` y después no le ponemos un código hexadecimal... depende; si es justo al principio de la cadena, por ejemplo:

```
unescape('%¿que saldra aqui?');
```
obtenemos la cadena vacía `''`, si ponemos

```
unescape('%a¿que saldra aqui?');
```
obtenemos `' que saldra aqui?'`, si ponemos

```
unescape('%a¿que s%aldra aqui?');
```
obtenemos `' que s dra aqui?'`, si ahora ponemos

```
unescape('%0f¿que saldra aqui?');
```
obtenemos `'|¿que saldra aqui?'`, si ponemos

```
unescape('¿que saldra %% aqui?');
```
obtenemos `'¿que saldra '`, ...

isNaN(valor);

Esta función evalua el argumento que se le pasa y devuelve `'true'` en el caso de que el argumento NO sea numérico. En caso contrario (i.e., el argumento pasado es numérico)

devuelve `'false'`. Por ejemplo:

```
isNaN(''Hola Mundo'');
```

devuelve `'true'`, pero

```
isNaN(''091'');
```

devuelve `'false'`.

Nota: Esta función **no funciona correctamente** con la versión **3.0** del navegador MSIE.

eval(expresion);

Esta función devuelve el resultado de evaluar la expresión pasada como parámetro. Veamos algunos ejemplos:

```
eval(''2+3'');
```

nos devuelve 5,

```
eval(2+3);
```

también nos devuelve 5,

`eval(''1.e-1*87/16'');` devuelve .5437500000000001, pero

```
eval(''Hola_Mundo'');
```

devuelve un error de JS por parte del navegador. Dice que `'Hola_Mundo'` no está definido. Esto es así porque la función `'eval'` espera una expresión que puede ser convertida a trozos más pequeños, números y operadores, para poder evaluarla. Para esta función, todo lo que no sea un número o un operador será considerado como una variable. Como no tenemos definida una variable global que se llame `Hola_Mundo`, es el navegador quien nos devuelve el error.

Vamos a ponernos en el caso en que sí tenemos definida esta variable global llamada `Hola_Mundo`. Supongamos que en la cabecera del documento, dentro de la directiva <SCRIPT>, tenemos la variable definida como:

```
var Hola_Mundo = ''Hola a todos'';
```

y ahora hacemos

```
eval('1+Hola_Mundo');
```

como `Hola_Mundo` no es número, piensa que es una variable, la busca, y ve que es una cadena. ¿Os imaginais el resultado de este `'eval'`?

Como el + también es un operador de cadenas, interpreta que `'1'` y `'Hola a todos'` son cadenas, y realiza la operación +, es decir, las concatena, dando como resultado `'1Hola a todos' :-)`

Sin embargo, si ponemos otra operación, por ejemplo:

```
eval('1*Hola_Mundo');
```

ahora sí, este `'eval'` nos devuelve `NaN`.

Para poder proponer algún ejercicio, os anticipo de la existencia de una función, `'prompt'`, que sirve para pedir datos. Esta función tiene la forma:

prompt("Mensaje informativo", "Respuesta por defecto"); La respuesta por defecto es optativa, si no la ponemos, no hay respuesta por defecto. Esta función devuelve una cadena con la respuesta que hayamos introducido, tras pulsar el botón 'Aceptar'.

Con esto, vamos a por un ejercicio.

Ejercicio: Usando la función `prompt`, pedir al usuario que introduzca algún tipo de entrada. Si la entrada es de tipo numérico, o su evaluación puede dar lugar a un número, que escriba en el documento ese número, y si no es de tipo numérico (y su evaluación tampoco), que diga que no puede evaluarse y que escriba en el documento la respuesta introducida por el usuario.

4 Introducción a los objetos

En el capítulo anterior describimos las funciones propias del lenguaje, funciones que no están ligadas a ningún objeto. Pero, ¿y qué son los objetos? De eso trata este capítulo.

4.1 Introducción a los objetos

En cualquier curso sobre programación estructurada se presentan unas convenciones a la hora de escribir algoritmos, unos ciertos tipos básicos de variables, unas estructuras de control, y cómo, a partir de estos elementos, construir estructuras más complejas para poder dar solución a problemas diversos como puede ser la gestión de un hospital u otros. Todo ello, desde un punto de vista más o menos secuencial y teniendo a las funciones y procedimientos como principales protagonistas del desarrollo de un programa, sin estar por ellos interrelacionadas con los datos que manejan.

Con la programación orientada a objetos (POO u OOP, a partir de ahora) este concepto cambia radicalmente, y el desarrollo de los programas se centran, como su nombre indica, en los objetos.

Y, ¿qué es un objeto?

La respuesta más llana que se puede dar es esta:

OBJETO = DATOS + FUNCIONES QUE TRABAJAN CON ESOS DATOS

La diferencia está en que esas funciones no están "aparte" de los datos, sino que forman parte de la misma estructura.

Vamos a ver un ejemplo: suponed que quereis hacer un programa de dibujo (ejemplo clásico). Es evidente que no lo vamos a hacer en modo texto (¡o eso creo! ;)), es más, probablemente querremos poner elementos como ventanas, botones, menús, barras de desplazamiento... y otros.

Pensando un poco, todo esto se puede llevar a cabo con lo que uno sabe de programación tradicional (de esto puedo dar fe). Sin embargo, ¿no sería más fácil (por ejemplo) tener un objeto VENTANA?

Este objeto podría tener las siguientes variables: coordenadas del rectángulo en el que se va a dibujar, estilo de dibujo... y podría tener unas funciones que, leyendo estos datos, dibujaran la ventana, cambiaran su tamaño, la cerraran...

La ventaja que tendría esto es que ya podemos crear cuantas ventanas queramos en nuestro programa, puesto que las funciones asociadas tendrían en cuenta las variables de este objeto y no tendríamos que pasárselas como argumentos.

Igual que con la ventana, se podría crear el objeto BOTON, que podría tener como

variables las coordenadas, si está pulsado o no, si está el ratón sobre él o no... y unas funciones que lo dibujaran, lo dibujaran pulsado, detectaran si se ha pulsado el botón, hicieran algo si ese botón se ha pulsado...

Las funciones de un objeto nos proporcionan un interfaz para manejarlo: no necesitamos saber cómo está hecho un objeto para poder utilizarlo.

Cuando creamos un objeto, en realidad estamos creando un 'molde', que recibe el nombre de clase, en el que especificamos todo lo que se puede hacer con los objetos (ahora sí) que utilicen ese molde. Es decir, en realidad lo que uno hace es crear una clase. Cuando va a utilizarla, crea instancias de la clase, y a estas instancias es a lo que se le llama objeto.

Por ejemplo, en el caso de la ventana, podría ser una cosa así:

```
CLASE Ventana
 VARIABLES
  x0, y0, x1, y1: ENTEROS
 FUNCIONES
  Inicializar_Coordenadas()
  Dibujar_Ventana()
  Mover_Ventana()
FIN CLASE Ventana
```

Y cuando queramos tener una (o varias) ventanas, las declararíamos (instanciaríamos) como cualquier otro tipo de variable:

```
Ventana Ventana_1, Ventana_2
```

A la hora de acceder a las variables que tiene cada una de estas ventanas (cuyos valores son distintos para cada ventana) o de utilizar las funciones de cada una de estas ventanas, habría que poner:

```
Ventana_1.x0 = 3
Ventana_2.Dibujar_Ventana()
```

es decir:

```
Objeto.Variable
Objeto.Funcion(argumentos)
```

Cuando se habla de POO se habla también de acceso a las partes de un objeto (partes públicas, privadas y protegidas), pero como JS es más simple, no tenemos distinción de accesos. Sin embargo, la forma de inicializar las coordenadas de la ventana más correcta no sería haciendo `Ventana_1.x0=3;` , etc., como he puesto en el ejemplo, sino que se encargará de ello una de sus propias funciones (una función especial llamada 'constructor'), pasándole como argumentos los valores de inicialización. En JS, el constructor es precisamente la función con la que crearemos el molde de la clase.

JS no es un lenguaje orientado a objetos propiamente dicho: está basado en unos objetos (los objetos del navegador y unos propios, sobre los que comenzaremos a hablar en la entrega siguiente) de los que podemos usar unas funciones que ya están definidas y cambiar los valores de unas variables, de acuerdo a la notación que hemos visto para acceder a las partes de un objeto. También disponemos de una notación en forma de array para acceder a cada una de las variables del objeto, esta es:

```
Objeto['variable']
```

Además, nos permite crear objetos, pero de una forma muy simple, puesto que no tiene una de las características esenciales de la POO: la herencia (entre otras).

Crear nuestros propios moldes para objetos será tan simple como hacer

```
function Mi_Objeto(dato_1, ..., dato_N)  {
  this.variable_1 = dato_1;
             ....
  this.variable_N = dato_N;

  this.funcion_1 = Funcion_1;
            ....
  this.funcion_N = Funcion_N;
  }
```

Según lo dicho unos párrafos más arriba, `Mi_Objeto` es el nombre del constructor de la clase. `'this'` es un objeto especial; direcciona el objeto actual que está siendo definido en la declaración. Es decir, se trata de una referencia al objeto actual (en este caso, el que se define). Al definir las funciones, sólo ponemos el nombre, no sus argumentos. La implementación de cada una de las funciones de la clase se hará de la misma forma que se declaran las funciones, es decir, haremos:

```
function Mi_Funcion([arg_1], ..., [arg_M])  {
  cosas varias que haga la funcion
  }
```

y cuando vayamos a crearlos, tenemos que usar un operador especial, `'new'`, seguido del constructor del objeto al que le pasamos como argumentos los argumentos con los que hemos definido el molde. Así, haciendo

```
var Un_Objeto = new Mi_OBjeto(dato1, ..., datoN);
```

creamos el objeto `'Un_Objeto'` según el molde `'Mi_Objeto'`, y ya podemos usar sus funciones y modificar o usar convenientemente sus atributos.

Con la versión 1.2 del lenguaje, también se pueden crear objetos de esta forma:

```
var Un_Objeto = { atributo1: valor1, ..., atributoN: valorN,
                  funcion1: Funcion1, ... , funcionN: FuncionN };
```

No hay lugar a confusión en cuanto a si el navegador sabe a qué clase pertenece el objeto, pues no puede haber dos clases que tengan una función que se llame igual y hagan cosas distintas; si no nos hemos dado cuenta y hemos definido dos veces la misma función, una para una clase y otra para otra clase, como no hay nada que distinga estas funciones, el navegador elige tomar la última de ellas que se haya definido con el mismo nombre. De lo que hay que asegurarse es de no usar el mismo nombre para funciones de dos clases distintas. JS no incorpora polimorfismo.

Como ejemplo de declaración de clases e instanciación de objetos, vamos a crear un molde para objetos de tipo círculo al que le pasamos el radio cuando lo inicialicemos, y que tenga una función que nos calcule su área, otra su longitud, y estos valores sean mostrados por pantalla.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 "http://www.w3.org/TR/html4/loose.dtd">
<HTML>
<HEAD>
 <TITLE> Ejemplo de creacion y uso de objetos </TITLE>
 <SCRIPT LANGUAGE="JavaScript">
 <!--
  var PI=3.14159265359;

  /* En primer lugar creamos la clase Circulo, cuyo constructor
     toma el mismo nombre y al que se le pasara un paramentro
     cuando instanciemos objetos */

  function Circulo(radio) {
    this.radio=radio;

    this.area=area;
    this.longitud=longitud;
    }

  /* Definimos las funciones que estaran asociadas a los objetos
     de tipo Circulo */

  function area() {
    var area = PI*this.radio*this.radio;
    alert("El area es: "+area);
    }

  function longitud() {
```

```
    var longitud = 2*PI*this.radio;
    alert("La longitud es: "+longitud);
    }

  /* Esta funcion sirve para probar la creacion de los objetos */

  function Crea_Dos_Circulos() {
    var circ1 = new Circulo(1);
    var circ2 = { radio: 2, area: area, longitud: longitud };

    circ1.area();
    circ1.longitud();
    circ2.area();
    circ2.longitud();
    circ1.radio=3;     // Cambiamos el valor del radio a circ1
    circ1.area();
    circ1.nombre="Soy el circulo 1";   // Anadimos un nuevo atributo
                                       // a circ1
    alert(circ1.nombre);
    alert(circ2.nombre);     // Y comprobamos que circ2 NO lo tiene
    }

 //-->
 </SCRIPT>
</HEAD>

<BODY onLoad="Crea_Dos_Circulos();"> </BODY>
</HTML>
```

(Nota: Ya veremos que no es necesario usar la variable PI, porque JS tiene su propio PI en alguna parte ;)).

Del ejemplo, lo único que cabe resaltar es que hemos añadido posteriormente un atributo a `'circ1'`, el atributo `'nombre'`, pero este atributo NO ha sido añadido a `'circ2'`. Cuando añadimos fuera de la clase atributos a un objeto, éstos son añadidos única y exclusivamente a ése objeto, pero a ningún otro de los objetos creados con la misma clase.

Ejercicio: Diseñad un objeto ventana con las siguientes características:

1. Sus variables serán las coordenadas de las esquinas del rectángulo que la define.
2. Sus métodos tendrán que hacer lo siguiente: Escribir las coordenadas de las esquinas en la ventana del navegador; poder cambiar estas coordenadas y volver a escribirlas para verificar el cambio.

Ejercicio: Diseñad un objeto triángulo con las siguientes características:

1. Sus variables serán las coordenadas de los vértices, la base y la altura. En principio se pueden pasar todas como argumentos iniciales.

2. Sus métodos permitirán calcular su área, escribirla en la ventana del navegador, cambiar las coordenadas de los vértices y los valores de la base y la altura.

3. Para los más atrevidos: JS incorpora un objeto llamado Math, que es el que tiene definidas las funciones matemáticas. Más adelante se hablará de este objeto. No obstante, si alguien se anima, que investigue un poco sobre él, y modifique el objeto triángulo de manera que sólo se le pase como argumentos las coordenadas del triángulo, definiendo una función que calcule a partir de dichas coordenadas la base y la altura del triángulo.

5 Los objetos del lenguaje. El objeto String

Una vez visto en la anterior entrega una introducción a la idea de la programación orientada a objetos, vamos a ver ahora qué nos proporciona JavaScript, pues tiene una librería básica de objetos que podemos instanciar y con los que podemos trabajar directamente.

Una primera clasificación del modelo de objetos lo dividiría en dos grandes grupos. Por una parte, tendríamos los objetos directamente relacionados con el navegador y las posibilidades de programación HTML (denominados, genéricamente, "objetos del navegador") y por otra parte un conjunto de objetos relacionados con la estructura del lenguaje, llamados genéricamente "objetos del lenguaje".

En las entregas de este capítulo vamos a ver este último gran grupo, el de los objetos del lenguaje. Estos objetos son los siguientes: String, Array, Math, Date, Boolean, Number y Function.

5.1 El objeto String

Este objeto nos permite hacer diversas manipulaciones con las cadenas, para que trabajar con ellas sea más sencillo. Cuando asignamos una cadena a una variable, JS está creando un objeto de tipo String que es el que nos permite hacer las manipulaciones.

5.2 Propiedades del objeto String

length Valor numérico que nos indica la longitud en caracteres de la cadena dada.

prototype Nos permite asignar nuevas propiedades al objeto String.

5.3 Métodos del objeto String

anchor(nombre) Crea un enlace asignando al atributo NAME el valor de 'nombre'. Este nombre debe estar entre comillas " "

big() Muestra la cadena de caracteres con una fuente grande.

blink() Muestra la cadena de texto con un efecto intermitente.

charAt(indice) Devuelve el carácter situado en la posición especificada por 'indice'.

fixed() Muestra la cadena de caracteres con una fuente proporcional.

fontcolor(color) Cambia el color con el que se muestra la cadena. La variable color debe ser especificada entre comillas: " ", o bien siguiendo el estilo de HTML, es decir "#RRGGBB" donde RR, GG, BB son los valores en hexadecimal para los colores

rojo, verde y azul, o bien puede ponerse un identificador válido de color entre comillas. Algunos de estos identificadores son "red", "blue", "yellow", "purple", "darkgray", "olive", "salmon", "black", "white", ...

fontsize(tamaño) Cambia el tamaño con el que se muestra la cadena. Los tamaños válidos son de 1 (más pequeño) a 7 (más grande).

indexOf(cadena_buscada,indice) Devuelve la posición de la primera ocurrencia de 'cadena_buscada' dentro de la cadena actual, a partir de la posición dada por 'indice'. Este último argumento es opcional y, si se omite, la busqueda comienza por el primer carácter de la cadena.

italics() Muestra la cadena en cursiva.

lastIndexOf(cadena_buscada,indice) Devuelve la posición de la última ocurrencia de 'cadena_buscada' dentro de la cadena actual, a partir de la posición dada por 'indice', y buscando hacia atrás. Este último argumento es opcional y, si se omite, la busqueda comienza por el último carácter de la cadena.

link(URL) Convierte la cadena en un vínculo asignando al atributo HREF el valor de URL.

small() Muestra la cadena con una fuente pequeña.

split(separador) Parte la cadena en un array de caracteres. Si el carácter separador no se encuentra, devuelve un array con un sólo elemento que coincide con la cadena original. A partir de NS 3, IE 4 (JS 1.2).

strike() Muestra la cadena de caracteres tachada.

sub() Muestra la cadena con formato de subíndice.

substring(primer_Indice,segundo_Indice) Devuelve la subcadena que comienza en la posición 'primer_Indice + 1' y que finaliza en la posición 'segundo_Indice'. Si 'primer_Indice' es mayor que 'segundo_Indice', empieza por 'segundo_Indice + 1' y termina en 'primer_Indice'. Si hacemos las cuentas a partir de 0, entonces es la cadena que comienza en 'primer_Indice' y termina en 'segundo_Indice - 1' (o bien 'segundo_Indice' y 'primer_Indice - 1' si el primero es mayor que el segundo).

sup() Muestra la cadena con formato de superíndice.

toLowerCase() Devuelve la cadena en minúsculas.

toUpperCase() Devuelve la cadena en minúsculas.

A continuación, vamos a ver un ejemplo en el que se muestran algunas de estas funciones:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 "http://www.w3.org/TR/html4/loose.dtd">
<HTML>
<HEAD>
 <TITLE> Ejemplo JS </TITLE>
 <SCRIPT LANGUAGE="JavaScript">
 <!--
 function a()  {
  var cad = "Hello World",i;
  var ja = new Array();

  ja = cad.split("o");

  with(document) {
   write("La cadena es: "+cad+"<BR>");
   write("Longitud de la cadena: "+cad.length+"<BR>");
   write("Haciendola ancla: "+cad.anchor("b")+"<BR>");
   write("En grande: "+cad.big()+"<BR>");
   write("Parpadea: "+cad.blink()+"<BR>");
   write("Caracter 3 es: "+cad.charAt(3)+"<BR>");
   write("Fuente FIXED: "+cad.fixed()+"<BR>");
   write("De color: "+cad.fontcolor("#FF0000")+"<BR>");
   write("De color: "+cad.fontcolor("salmon")+"<BR>");
   write("Tamano 7: "+cad.fontsize(7)+"<BR>");
   write("<I>orl</I> esta en la posicion: "+cad.indexOf("orl"));
   write("<BR>En cursiva: "+cad.italics()+"<BR>");
   write("La primera <I>l</I> esta, empezando a contar por detras,");
   write(" en la posicion: "+cad.lastIndexOf("l")+"<BR>");
   write("Haciendola enlace: "+cad.link("doc.htm")+"<BR>");
   write("En pequeno: "+cad.small()+"<BR>");
   write("Tachada: "+cad.strike()+"<BR>");
   write("Subindice: "+cad.sub()+"<BR>");
   write("Superindice: "+cad.sup()+"<BR>");
   write("Minusculas: "+cad.toLowerCase()+"<BR>");
   write("Mayusculas: "+cad.toUpperCase()+"<BR>");
   write("Subcadena entre los caracteres 3 y 10: ");
   write(cad.substring(2,10)+"<BR>");
   write("Entre los caracteres 10 y 3: "+cad.substring(10,2)+"<BR>");
   write("Subcadenas resultantes de separar por las <B>o:</B><BR>");
   for(i=0;i<ja.length;i++) write(ja[i]+"<BR>");
   }
 }
 //-->
 </SCRIPT>
</HEAD>
```

```
<BODY>
 <SCRIPT LANGUAGE="JavaScript">
 <!--
  a();
 //-->
 </SCRIPT>
</BODY>
</HTML>
```

Seguimos en el próximo capítulo con otros dos objetos importantes dentro de los objetos del lenguaje: Array y Math. El primero nos va a permitir trabajar con vectores (y matrices) de la forma usual, dándonos algunas facilidades extra, y el segundo nos permitirá hacer los cálculos que necesitemos en nuestro programa.

6 Los objetos del lenguaje. Los objetos Array y Math

6.1 El objeto Array

Este objeto nos va a dar la facilidad de construir arrays cuyos elementos pueden contener cualquier tipo básico, y cuya longitud se modificará de forma dinámica siempre que añadamos un nuevo elemento (y, por tanto, no tendremos que preocuparnos de esa tarea). Para poder tener un objeto array, tendremos que crearlo con su constructor, por ejemplo, si escribimos:

```
a=new Array(15);
```

tendremos creada una variable 'a' que contendrá 15 elementos, enumerados del 0 al 14. Para acceder a cada elemento individual usaremos la notación a[i], donde i variará entre 0 y N-1, siendo N el número de elementos que le pasamos al constructor.

También podemos inicializar el array a la vez que lo declaramos, pasando los valores que queramos directamente al constructor, por ejemplo:

```
a=new Array(21,''cadena'',true);
```

que nos muestra, además, que los elementos del array no tienen por qué ser del mismo tipo.

Por tanto: si ponemos un argumento al llamar al constructor, este será el número de elementos del array (y habrá que asignarles valores posteriormente), y si ponemos más de uno, será la forma de inicializar el array con tantos elementos como argumentos reciba el constructor.

Podríamos poner como mención especial de esto lo siguiente. Las inicializaciones que vemos a continuación:

```
a=new Array("cadena");
a=new Array(false);
```

Inicializan el array 'a', en el primer caso, con un elemento cuyo contenido es la cadena 'cadena', y en el segundo caso con un elemento cuyo contenido es 'false'.

Lo comentado anteriormente sobre inicialización de arrays con varios valores, significa que si escribimos

```
a=new Array(2,3);
```

NO vamos a tener un array con 2 filas y 3 columnas, sino un array cuyo primer elemento será el 2 y cuyo segundo elemento será el 3. Entonces, ¿cómo creamos un array bidimensional? (un array bidimensional es una construcción bastante frecuente). Creando un array con las filas deseadas y, después, cada elemento del array se inicializará con un array con

las columnas deseadas. Por ejemplo, si queremos crear un array con 4 filas y 7 columnas, bastará escribir:

```
a=new Array(4);
for(i=0;i<4;i++)  a[i]=new Array(7);
```

y para referenciar al elemento que ocupa la posición (i,j), escribiremos `a[i][j]`;

Vistas las peculiaridades de la creación de arrays, vamos ahora a estudiar sus propiedades y sus métodos.

6.1.1 Propiedades del objeto Array

length Esta propiedad nos dice en cada momento la longitud del array, es decir, cuántos elementos tiene.

prototype

6.1.2 Métodos del objeto Array

join(separador) Une los elementos de las cadenas de caracteres de cada elemento de un array en un string, separando cada cadena por el separador especificado.

reverse() Invierte el orden de los elementos del array.

sort() Ordena los elementos del array siguiendo el orden lexicográfico

Veamos un ejemplo que nos muestra todas estas posibilidades:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<HTML>
<HEAD>
 <TITLE> Probando el objeto Array </TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!--
function a() {
  var j=new Array(2),h=new Array(1), i=new Array(1,"Hola",3);
  var b=new Array("Palabra","Letra","Amor","Color","Carino");
  var c=new Array("Otra cadena con palabras");
  var d=new Array(false);

  j[0]=new Array(3);
```

```
  j[1]=new Array(2);

  j[0][0]=0; j[0][1]=1; j[0][2]=2;
  j[1][0]=3; j[1][1]=4; j[1][2]=5;

  document.write(c);
  document.write("<P>"+d+"<P>");

  document.write("j[0][0]="+j[0][0]+"; j[0][1]="+j[0][1]+
                 "; j[0][2]="+j[0][2]+"<BR>");
  document.write("j[1][0]="+j[1][0]+"; j[1][1]="+j[1][1]+
                 "; j[1][2]="+j[1][2]);
  document.write("<P>h= "+(h[0]='Hola')+"<P>");
  document.write("i[0]="+i[0]+"; i[1]="+i[1]+"; i[2]="+i[2]+"<P>");
  document.write("Antes de ordenar: "+b.join(', ')+"<P>");
  document.write("Ordenados: "+b.sort()+"<P>");
  document.write("Ordenados en orden inverso: "+b.sort().reverse());
  }
//-->
</SCRIPT></HEAD>

<BODY>
<SCRIPT LANGUAGE="JavaScript">
<!--
  a();
//-->
</SCRIPT>
</BODY>
</HTML>
```

Y esto es todo en cuanto al objeto Array. A partir de aquí, todo depende de cómo necesitemos usarlo.

6.2 El objeto Math

Este objeto se utiliza para poder realizar cálculos en nuestros scripts. Tiene la peculiaridad de que sus propiedades no pueden modificarse, sólo consultarse. Estas propiedades son constantes matemáticas de uso frecuente en algunas tareas, por ello es lógico que sólo pueda consultarse su valor pero no modificarlo.

6.2.1 Propiedades del objeto Math

E Numero 'e', base de los logaritmos naturales (neperianos).

LN2 Logaritmo neperiano de 2.

LN10 Logaritmo neperiano de 10.

LOG2E Logaritmo en base 2 de e.

LOG10E Logaritmo en base 10 de e.

PI Numero PI.

SQRT1_2 Raíz cuadrada de $\frac{1}{2}$.

SQRT2 Raíz cuadrada de 2.

6.2.2 Métodos del objeto Math

abs(numero) Función valor absoluto.

acos(numero) Función arcocoseno. Devuelve un valor cuyas unidades son radianes o NaN. 'numero' debe pertenecer al rango $[-1, 1]$, en otro caso devuelve NaN.

asin(numero) Función arcoseno. Devuelve un valor cuyas unidades son radianes o NaN. 'numero' debe pertenecer al rango $[-1, 1]$, en otro caso devuelve NaN.

atan(numero) Función arcotangente. Devuelve un valor cuyas unidades son radianes o NaN.

atan2(x,y) Devuelve el ángulo formado por el vector de coordenadas (x,y) con respecto al eje OX.

ceil(numero) Devuelve el entero obtenido de redondear 'numero' "por arriba".

cos(numero) Devuelve el coseno de 'numero' (que debe estar en radianes) o NaN.

exp(numero) Devuelve el valor e^{numero}.

floor(numero) Devuelve el entero obtenido de redondear 'numero' "por abajo".

log(numero) Devuelve el logaritmo neperiano de 'numero'.

max(x,y) Devuelve el máximo de 'x' e 'y'.

min(x,y) Devuelve el mínimo de 'x' e 'y'.

pow(base,exp) Devuelve el valor $base^{exp}$.

random() Devuelve un numero pseudoaleatorio entre 0 y 1.

round(numero) Redondea 'numero' al entero más cercano.

sin(numero) Devuelve el seno de 'numero' (que debe estar en radianes) o NaN.

sqrt(numero) Devuelve la raíz cuadrada de 'numero'.

tan(numero) Devuelve la tangente de 'numero' (que debe estar en radianes) o NaN.

Aunque creo que es obvio, pero añado que, para poder usar alguna de sus propiedades o de sus métodos, tendremos que anteponer la palabra 'Math', por ejemplo, si queremos calcular el seno de PI, tendremos que escribir Math.sin(Math.PI)

Nos vemos en el próximo capítulo, en el que terminaremos con los objetos del lenguaje (ya habrá ganas, ¿no? ;-))

7 Los objetos del lenguaje. Los objetos Date, Boolean, Number y Function

7.1 El objeto Date

Este objeto nos va a permitir hacer manipulaciones con fechas: poner fechas, consultarlas... para ello, debemos saber lo siguiente: JS maneja fechas en milisegundos. Los meses de Enero a Diciembre vienen dados por un entero cuyo rango varía entre el 0 y el 11 (es decir, el mes 0 es Enero, el mes 1 es Febrero, y así sucesivamente), los días de la semana de Domingo a Sábado vienen dados por un entero cuyo rango varía entre 0 y 6 (el día 0 es el Domingo, el día 1 es el Lunes, ...), los años se ponen tal cual, y las horas se especifican con el formato HH:MM:SS. Podemos crear un objeto Date vacío, o podemos crealo dándole una fecha concreta. Si no le damos una fecha concreta, se creará con la fecha correspondiente al momento actual en el que se crea. Para crearlo dándole un valor, tenemos estas posibilidades:

```
var Mi_Fecha = new Date(año, mes);
var Mi_Fecha = new Date(año, mes, día);
var Mi_Fecha = new Date(año, mes, día, horas);
var Mi_Fecha = new Date(año, mes, día, horas, minutos);
var Mi_Fecha = new Date(año, mes, día, horas, minutos, segundos);
```

En 'día' pondremos un número del 1 al máximo de días del mes que toque. Todos los valores que tenemos que pasar al constructor son enteros. Pasamos a continuación a estudiar los métodos de este objeto.

7.1.1 Métodos del objeto Date

getDate(); Devuelve el día del mes actual como un entero entre 1 y 31.

getDay(); Devuelve el día de la semana actual como un entero entre 0 y 6.

getHours(); Devuelve la hora del día actual como un entero entre 0 y 23.

getMinutes(); Devuelve los minutos de la hora actual como un entero entre 0 y 59.

getMonth(); Devuelve el mes del año actual como un entero entre 0 y 11.

getSeconds(); Devuelve los segundos del minuto actual como un entero entre 0 y 59.

getTime(); Devuelve el tiempo transcurrido en milisegundos desde el 1 de enero de 1970 hasta el momento actual.

getYear(); Devuelve el año actual como un entero.

setDate(día_mes); Pone el día del mes actual en el objeto Date que estemos usando.

setDay(día_semana); Pone el día de la semana actual en el objeto Date que estemos usando.

setHours(horas); Pone la hora del día actual en el objeto Date que estemos usando.

setMinutes(minutos); Pone los minutos de la hora actual en el objeto Date que estemos usando.

setMonth(mes); Pone el mes del año actual en el objeto Date que estemos usando.

setSeconds(segundos); Pone los segundos del minuto actual en el objeto Date que estemos usando.

setTime(milisegundos); Pone la fecha que dista los milisegundos que le pasemos del 1 de enero de 1970 en el objeto Date que estemos usando.

setYear(año); Pone el año actual en el objeto Date que estemos usando.

toGMTString(); Devuelve una cadena que usa las convenciones de Internet con la zona horaria GMT.

7.2 El objeto Boolean

Este objeto nos permite crear booleanos, esto es, un tipo de dato que es cierto o falso, tomando los valores 'true' o 'false'. Podemos crear objetos de este tipo mediante su constructor. Veamos varios ejemplos:

`a = new Boolean();` asigna a 'a' el valor 'false'
`a = new Boolean(0);` asigna a 'a' el valor 'false'
`a = new Boolean("");` asigna a 'a' el valor 'false'
`a = new Boolean(false);` asigna a 'a' el valor 'false'
`a = new Boolean(`*numero_distinto_de_0*`);` asigna a 'a' el valor 'true'
`a = new Boolean(true);` asigna a 'a' el valor 'true'

Por medio de la propiedad prototype podemos darle más propiedades.

7.3 El objeto Number

Este objeto representa el tipo de dato 'número' con el que JS trabaja. Podemos asignar a una variable un número, o podemos darle valor, mediante el constructor Number, de esta forma:

`a = new Number(valor);`, por ejemplo, `a = new Number(3.2);` da a 'a' el valor 3.2. Si no pasamos algún valor al constructor, la variable se inicializará con el valor 0.

Este objeto cuenta con varias propiedades, sólo accesibles desde Number (ahora vemos qué quiero decir):

MAX_VALUE Valor máximo que se puede manejar con un tipo numérico

MIN_VALUE Valor mínimo que se puede manejar con un tipo numérico

NaN Representación de un dato que no es un número

NEGATIVE_INFINITY Representación del valor a partir del cual hay desbordamiento negativo (underflow)

POSITIVE_INFINITY Representación del valor a partir del cual hay desbordamiento positivo (overflow)

Para consultar estos valores, no podemos hacer:

```
a = new Number();
alert(a.MAX_VALUE);
```

porque JS nos dirá 'undefined', tenemos que hacerlo directamente sobre Number, es decir, tendremos que consultar los valores que hay en Number.MAX_VALUE, Number.MIN_VALUE, etc.

7.4 El objeto Function

Ya hablamos de este objeto, sin mencionar que era un objeto, cuando hablamos de declaración de funciones. Nos proporciona la propiedad 'arguments' que, como ya sabemos, es un Array con los argumentos que se han pasado al llamar a una función. Por el hecho de ser un Array, cuenta con todas las propiedades y los métodos de estos objetos.

7.5 Unas consideraciones finales

No sé si os habreis dado cuenta, pero estos objetos propios del lenguaje son (salvo 'Math' y 'Date'), precisamente, los ¨tipos de datos¨ con los que cuenta. Lo que en realidad sucede es esto: al crear una variable, en cuanto sabe de qué tipo es, CREA un objeto de ese tipo para la variable en cuestión. Por eso, si tenemos una variable que sea una cadena, automáticamente podemos usar los métodos del objeto String, porque lo que hemos hecho ha sido crear un objeto String para esa variable. En cuanto le asignamos otro tipo (por ejemplo, un número), destruye ese objeto y creo otro del tipo de lo que le hayamos asignado a la variable (si es un número, entonces sería un objeto de tipo Number), con lo que podremos usar los métodos y las propiedades de ese objeto.

8 Los objetos del navegador. Jerarquía

En este capítulo vamos a estudiar la jerarquía que presentan los objetos del navegador, atendiendo a una relación contenedor - contenido que se da entre estos objetos. De forma esquemática, esta jerarquía podemos representarla de esta manera (al lado está la directiva HTML con que se incluyen en el documento objetos de este tipo, cuando exista esta directiva):

```
* window
  + history
  + location
  + document   <BODY> ... </BODY>
     - anchor  <A NAME="..."> ... </A>
     - applet  <APPLET> ... </APPLET>
     - area    <MAP> ... </MAP>
     - form    <FORM> ... </FORM>
        + button     <INPUT TYPE="button">
        + checkbox   <INPUT TYPE="checkbox">
        + fileUpload <INPUT TYPE="file">
        + hidden     <INPUT TYPE="hidden">
        + password   <INPUT TYPE="password">
        + radio      <INPUT TYPE="radio">
        + reset      <INPUT TYPE="reset">
        + select     <SELECT> ... </SELECT>
           - options <INPUT TYPE="option">
        + submit     <INPUT TYPE="submit">
        + text       <INPUT TYPE="text">
        + textarea   <TEXTAREA> ... </TEXTAREA>
     - image   <IMG SRC="...">
     - link    <A HREF="..."> ... </A>
     - plugin  <EMBED SRC="...">
  + frame      <FRAME>
* navigator
```

Según esta jerarquía, podemos entender el objeto `'area'` (por poner un ejemplo) como un objeto dentro del objeto `'document'` que a su vez está dentro del objeto `'window'`. Hay que decir que la notación '.' también se usa para denotar a un objeto que está dentro de un objeto.

Por ejemplo, si queremos hacer referencia a una caja de texto, tendremos que escribir

```
ventana.documento.formulario.caja_de_texto
```

donde `'ventana'` es el nombre del objeto window (su nombre por defecto es window), `'documento'` es el nombre del objeto document (cuyo nombre por defecto es docu-

ment), `'formulario'` es el nombre del objeto forms (veremos que forms es un array) y `'caja_de_texto'` es el nombre del objeto textarea (cuyo nombre por defecto es textarea).

En la mayoría de los casos podemos ignorar la referencia a la ventana actual (window), pero será necesaria esta referencia cuando estemos utilizando múltiples ventanas, o cuando usemos frames. Cuando estemos usando un único frame, podemos pues ignorar explícitamente la referencia al objeto window, ya que JS asumirá que la referencia es de la ventana actual.

También podemos utilizar la notación de array para referirnos a algún objeto, por ejemplo, cuando los objetos a usar no tienen nombre, como en este caso:

```
document.forms[0].elements[1];
```

hace referencia al segundo elemento del primer formulario del documento; este elemento será el segundo que se haya creado en la página HTML.

Capítulo breve pero denso (aunque no lo parezca); conviene comprender bien la jerarquía de los objetos del navegador y las relaciones que se dan entre ellos. En el próximo capítulo comenzamos a describir estos objetos.

9 Los objetos del navegador. El objeto window

Con esta entrega comienza la descripción de las propiedades y los métodos de los objetos del navegador. No es mi intención hacer una descripción exhaustiva de todas y cada una de las propiedades y métodos, objeto por objeto, con todo detalle. Mi intención es hacer una descripción más o menos detallada de las propiedades y métodos que tienen más posibilidad de ser usados. Es decir, que si me dejo alguna propiedad y/o método por comentar, siempre podeis buscarla en alguna parte de la red. El lenguaje está sujeto a constantes revisiones, por lo que es muy probable que me quede obsoleta antes de terminar con el curso %-)

9.1 El objeto window

Se trata del objeto más alto en la jerarquía del navegador (navigator es un objeto independiente de todos en la jerarquía), pues todos los componentes de una página web están situados dentro de una ventana. El objeto window hace referencia a la ventana actual. Veamos a continuación sus propiedades y sus métodos.

9.1.1 Propiedades del objeto window

closed Válida a partir de JS 1.1 (Netscape 3 en adelante y MSIE 4 en adelante). 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.

frames Es un array: cada elemento de este array (frames[0], frames[1], ...) es uno de los frames que contiene la ventana. Su orden se asigna según se definen en el documento HTML.

history Se trata de un array que representa las URLS visitadas por la ventana (están almacenadas en su historial).

length Variable que nos indica cuántos frames tiene la ventana actual.

location Cadena con la URL de la barra de dirección.

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 usando el método open() que veremos cuando estudiemos los métodos.

parent Referencia al objeto window que contiene el frameset.

self Es un nombre alternativo del window actual.

status String con el mensaje que tiene la barra de estado.

top Nombre alternativo de la ventana del nivel superior.

window Igual que self: nombre alternativo del objeto window actual.

9.1.2 Métodos del objeto window

alert(mensaje) Muestra el mensaje 'mensaje' en un cuadro de diálogo

blur() Elimina el foco del objeto window actual. A partir de NS 3, IE 4 (JS 1.1).

clearInterval(id) Elimina el intervalo referenciado por 'id' (ver el método setInterval(), también del objeto window). A partir de NS 4, IE 4 (JS 1.2).

clearTimeout(nombre) Cancela el intervalo referenciado por 'nombre' (ver el método setTimeout(), también del objeto window).

close() Cierra el objeto window actual.

confirm(mensaje) Muestra un cuadro de diálogo con el mensaje 'mensaje' y dos botones, uno de aceptar y otro de cancelar. Devuelve true si se pulsa aceptar y devuelve false si se pulsa cancelar.

focus() Captura el foco del ratón sobre el objeto window actual. A partir de NS 3, IE 4 (JS 1.1).

moveBy(x,y) Mueve el objeto window actual el número de pixels especificados por (x,y). A partir de NS 4 (JS 1.2).

moveTo(x,y) Mueve el objeto window actual a las coordenadas (x,y). A partir de NS 4 (JS 1.2).

open(URL,nombre,caracteristicas) Abre la URL que le pasemos como primer parámetro en una ventana de nombre 'nombre'. Si esta ventana no existe, abrirá una ventana nueva en la que mostrará el contenido con las características especificadas. Las características que podemos elegir para la ventana que queramos abrir son las siguientes:

- toolbar = [yes|no|1|0]
  Nos dice si la ventana tendrá barra de herramientas (yes,1) o no la tendrá (no,0).
- location = [yes|no|1|0]
  Nos dice si la ventana tendrá campo de localización o no.
- directories = [yes|no|1|0]
  Nos dice si la nueva ventana tendrá botones de dirección o no.
- status = [yes|no|1|0]
  Nos dice si la nueva ventana tendrá barra de estado o no.
- menubar = [yes|no|1|0]
  Nos dice si la nueva ventana tendrá barra de menús o no.

- scrollbars = [yes|no|1|0]
  Nos dice si la nueva ventana tendrá barras de desplazamiento o no.
- resizable = [yes|no|1|0]
  Nos dice si la nueva ventana podrá ser cambiada de tamaño (con el ratón) o no.
- width = px
  Nos dice el ancho de la ventana en pixels.
- height = px
  Nos dice el alto de la ventana en pixels.
- outerWidth = px
  Nos dice el ancho *total* de la ventana en pixels. A partir de NS 4 (JS 1.2).
- outerHeight = px
  Nos dice el alto *total* de la ventana el pixels. A partir de NS 4 (JS 1.2)
- left = px
  Nos dice la distancia en pixels desde el lado izquierdo de la pantalla a la que se debe colocar la ventana.
- top = px
  Nos dice la distancia en pixels desde el lado superior de la pantalla a la que se debe colocar la ventana.

Activar o desactivar una característica de la ventana, automáticamente desactiva todas las demás.

prompt(mensaje,respuesta_por_defecto) Muestra un cuadro de diálogo que contiene una caja de texto en la cual podremos escribir una respuesta a lo que nos pregunte en 'mensaje'. El parámetro 'respuesta_por_defecto' es opcional, y mostrará la respuesta por defecto indicada al abrirse el cuadro de diálogo. El método retorna una cadena de caracteres con la respuesta introducida.

scroll(x,y) Desplaza el objeto window actual a las coordenadas especificadas por (x,y). A partir de NS3, IE4 (JS 1.1).

scrollBy(x,y) Desplaza el objeto window actual el número de pixels especificado por (x,y). A partir de NS4 (JS 1.2).

scrollTo(x,y) Desplaza el objeto window actual a las coordenadas especificadas por (x,y). A partir de NS4 (JS 1.2).

setInterval(expresion,tiempo) Evalua 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(). A partir de NS4, IE4 (JS 1.2).

setTimeout(expresion,tiempo) Evalua 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(). A partir de NS4, IE4 (JS 1.2).

Por mencionar algunos...

Me dejo en el tintero otras propiedades y métodos como innerHeight, innerWidth, outerHeight, outerWidth, pageXOffset, pageYOffset, personalbar, scrollbars, back(), find(["cadena"],[caso,bkwd]), forward(), home(), print(), stop()... todas ellas disponibles a partir de NS 4 (JS 1.2) y cuya explicación remito como ejercicio al lector interesado en saber más sobre el objeto window.

Para finalizar, veamos un pequeño ejemplito cuyo resultado podreis apreciar si pulsais el botón que hay a continuación del ejemplo:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 "http://www.w3.org/TR/html4/loose.dtd">
<HTML>
<HEAD>
 <TITLE> Ejemplo JS </TITLE>
 <SCRIPT LANGUAGE="JavaScript">
 <!--
 function a()  {
   var opciones="left=100,top=100,width=250,height=150";

   mi_ventana = window.open("","",opciones);
   mi_ventana.moveBy(100,100);
   mi_ventana.moveTo(400,100);
   mi_ventana.document.write("Una prueba de abrir ventanas");
   }
 //-->
 </SCRIPT>
</HEAD>

<BODY>
<SCRIPT LANGUAGE="JavaScript">
<!--
 a();
//-->
</SCRIPT>
</BODY>
</HTML>
```

Y eso es todo, en el siguiente capítulo estudiaremos cuatro objetos más; frame, location, history y navigator.

10 Los objetos del navegador. Los objetos frame, location, history y navigator

A partir de ahora será más rápido explicar las propiedades y los métodos de algunos objetos, puesto que van a ser los mismos que tenía el objeto window, pero en vez de referidos a window serán referidos a ellos. Es por ello que en esta entrega hay espacio suficiente para estudiar hasta cuatro objetos de una vez :-)

10.1 El objeto frame

Todos sabemos que la ventana del navegador puede ser dividida en varios frames que contengan cada uno de ellos un documento en el que mostrar contenidos diferentes. Al igual que con las ventanas, cada uno de estos frames puede ser nombrado y referenciado, lo que nos permite cargar documentos en un marco sin que esto afecte al resto.

10.1.1 Propiedades del objeto frame

frames Array que representa los objetos frame que están en el window actual. Su orden de aparición (frame[0],...) es el orden en que son definidos en el documento HTML.

parent

self

top

window

10.1.2 Métodos del objeto frame

alert(mensaje)

blur()

clearInterval(id)

clearTimeout(nombre)

close()

confirm(mensaje)

focus()

moveBy(x,y)

moveTo(x,y)

open(URL,nombre,caracteristicas) Caracteristicas:

```
toolbar = [yes|no|1|0], location = [yes|no|1|0],
directories = [yes|no|1|0], status = [yes|no|1|0],
menubar = [yes|no|1|0], scrollbars = [yes|no|1|0],
resizable = [yes|no|1|0], width = px, height = px
```

prompt(mensaje,resp_defecto)

scroll(x,y)

setInterval(expresion,tiempo)

setTimeout(expresion,tiempo)

Como ejercicio, podeis crear una ventana con tres frames: dos filas que dividan la pantalla en una subpantalla superior y otra inferior, y esta subpantalla inferior a su vez dividida en dos frames columna.

Haced una función que al cargarse el documento escriba el nombre del frame (que lo asignábamos con el parámetro NAME de la directiva FRAME) en su frame correspondiente.

10.2 El objeto location

Este objeto contiene la URL actual así como algunos datos de interés respecto a esta URL. Su finalidad principal es, por una parte, modificar el objeto location para cambiar a una nueva URL, y extraer los componentes de dicha URL de forma separada para poder trabajar con ellos de forma individual si es el caso. Recordemos que la sintaxis de una URL era:

```
protocolo://maquina_host[:puerto]/camino_al_recurso
```

10.2.1 Propiedades del objeto location

hash Cadena que contiene el nombre del enlace, dentro de la URL.

host Cadena que contiene el nombre del servidor y el número del puerto, dentro de la URL.

hostname Cadena que contiene el nombre de dominio del servidor (o la dirección IP), dentro de la URL.

href Cadena que contiene la URL completa.

pathname Cadena que contiene el camino al recurso, dentro de la URL.

port Cadena que contiene el número de puerto del servidor, dentro de la URL.

protocol Cadena que contiene el protocolo utilizado (incluyendo los dos puntos), dentro de la URL.

search Cadena que contiene la información pasada en una llamada a un script CGI, dentro de la URL.

10.2.2 Métodos del objeto location

reload() Vuelve a cargar la URL especificada en la propiedad href del objeto location.

replace(cadenaURL) Reemplaza el historial actual mientras carga la URL especificada en cadenaURL.

10.3 El objeto history

Este objeto se encarga de almacenar una lista con los sitios por los que se ha estado navegando, es decir, guarda las referencias de los lugares visitados. Se utiliza, sobre todo, para movernos hacia delante o hacia atrás en dicha lista.

10.3.1 Propiedades del objeto history

current Cadena que contiene la URL completa de la entrada actual en el historial.

next Cadena que contiene la URL completa de la siguiente entrada en el historial.

length Entero que contiene el número de entradas del historial (i.e., cuántas direcciones han sido visitadas).

previous Cadena que contiene la URL completa de la anterior entrada en el historial.

10.3.2 Métodos del objeto history

back() Vuelve a cargar la URL del documento anterior dentro del historial.

forward() Vuelve a cargar la URL del documento siguiente dentro del historial.

go(posicion) Vuelve a cargar la URL del documento especificado por 'posicion' dentro del historial. 'posicion' puede ser un entero, en cuyo caso indica la posición relativa del documento dentro del historial; o puede ser una cadena de caracteres, en cuyo caso representa toda o parte de una URL que esté en el historial.

10.4 El objeto navigator

Este objeto simplemente nos da información relativa al navegador que esté utilizando el usuario.

NOTA: Este objeto está contenido en el objeto window en Explorer a partir de la versión 4. Esto significa que, para saber si estamos con este navegador, tendremos que comprobar la existencia de navigator como "hijo" de window, simplemente escribiendo `if(window.navigator) { ... }`

10.4.1 Propiedades del objeto navigator

appCodeName Cadena que contiene el nombre del código del cliente.

appName Cadena que contiene el nombre del cliente.

appVersion Cadena que contiene información sobre la versión del cliente.

language Cadena de dos caracteres que contiene información sobre el idioma de la versión del cliente.

mimeTypes Array que contiene todos los tipos MIME soportados por el cliente. A partir de NS 3 (JS 1.1).

platform Cadena con la plataforma sobre la que se está ejecutando el programa cliente.

plugins Array que contiene todos los plug-ins soportados por el cliente. A partir de NS 3 (JS 1.1).

userAgent Cadena que contiene la cabecera completa del agente enviada en una petición HTTP. Contiene la información de las propiedades appCodeName y appVersion.

10.4.2 Métodos del objeto navigator

javaEnabled() Devuelve 'true' si el cliente permite la utilización de Java, en caso contrario, devuelve 'false'.

Como ejercicio, podeis hacer una función JS que imprima en la ventana del navegador toda la información relativa al mismo, consultando para ello el valor en cada una de las propiedades del objeto correspondiente.

11 Los objetos del navegador. Los objetos document, anchor, link e image

Continuamos con los objetos del navegador. De este capítulo, sin duda el más importante es el objeto document, que es el que vamos a estudiar primero.

11.1 El objeto document

El objeto document es el que tiene el contenido de toda la página que se está visualizando. Esto incluye el texto, imágenes, enlaces, formularios, ... Gracias a este objeto vamos a poder añadir dinámicamente contenido a la página, o hacer cambios, según nos convenga.

11.1.1 Propiedades del objeto document

alinkColor Esta propiedad tiene almacenado el color de los enlaces activos

anchors Se trata de un array con los enlaces internos existentes en el documento

applets Es un array con los applets existentes en el documento

bgColor Propiedad que almacena el color de fondo del documento

cookie Es una cadena con los valores de las cookies del documento actual

domain Guarda el nombre del servidor que ha servido el documento

embeds Es un array con todos los EMBED del documento

fgColor En esta propiedad tenemos el color del primer plano

forms Se trata de un array con todos los formularios del documento. Los formularios tienen a su vez elementos (cajas de texto, botones, etc) que tienen sus propias propiedades y métodos, y serán tratados en el siguiente capítulo.

images Array con todas las imágenes del documento

lastModified Es una cadena con la fecha de la última modificación del documento

linkColor Propiedad que almacena el color de los enlaces

links Es un array con los enlaces externos

location Cadena con la URL del documento actual

referrer Cadena con la URL del documento que llamó al actual, en caso de usar un enlace.

title Cadena con el título del documento actual

vlinkColor Propiedad en la que se guarda el color de los enlaces visitados

11.1.2 Métodos del objeto document

clear(); Limpia la ventana del documento

close(); Cierra la escritura sobre el documento actual

open(mime,"replace"); Abre la escritura sobre un documento. 'mime' es el tipo de documento soportado por el navegador. Si ponemos "replace", se reusa el documento anterior en el historial.

write(); Escribe texto en el documento.

writeln(); Escribe texto en el documento, y además lo finaliza con un salto de línea

De todos estos métodos, sin dudarlo el más usado es document.write(); ¿Por qué este precisamente? Vamos a responder a esta pregunta con un ejemplo: supongamos que en nuestra página queremos hacer una tabla tipo tabla con 10 columnas, 20 filas, de manera que en cada celda queremos escribir algo que sea consecutivo, por ejemplo, el valor de la suma de la fila en la que estamos y la columna. Este tipo de cosas son fácilmente programables y nos ahorraremos escribir bastante HTML para hacer esa tabla. Por ejemplo, la que hemos puesto como ejemplo, podría escribirse más brevemente que "a lo bruto" como sigue:

```
function HazTabla(fila,col)  {
  var i,j;

  document.write("<TABLE>\n");
  for(i=0;i<fila;i++)  {
    document.write("<TR>\n");
    for(j=0;j<col;j++)
      document.write("<TD>"+(i+j)+"</TD>\n");
    document.write("</TR>\n");
    }
  document.write("</TABLE>");
  }
```

Además, al hacer la función de forma que fila, col sean variables, nos ahorramos cambiar sus valores cada vez que los queramos modificar. Esto ilustra la utilidad del 'document.write' y por qué es uno de los métodos más usados.

11.2 El objeto link

11.2.1 Propiedades del objeto link

Este objeto engloba todas las propiedades que tienen los enlaces externos al documento actual.

target Es una cadena que tiene el nombre de la ventana o del frame especificado en el parámetro TARGET

hash Es una cadena con el nombre del enlace, dentro de la URL

host Es una cadena con el nombre del servidor y número de puerto, dentro de la URL

hostname Es una cadena con el nombre de dominio del servidor (o la dirección IP) dentro de la URL

href Es una cadena con la URL completa

pathname Es una cadena con el camino al recurso, dentro de la URL

port Es una cadena con el número de puerto, dentro de la URL

protocol Es una cadena con el protocolo usado, incluyendo los : (los dos puntos), dentro de la URL

search Es una cadena que tiene la información pasada en una llamada a un script CGI, dentro de la URL

11.3 El objeto anchor

Este objeto engloba todas las propiedades que tienen los enlaces internos al documento actual.

11.3.1 Propiedades del objeto anchor

href Es una cadena que contiene la URL completa

target Es una cadena que tiene el nombre de la ventana o del frame especificado en el parámetro TARGET

11.4 El objeto image

Gracias a este objeto (disponible a partir de la versión 1.1 de JavaScript, aunque Microsoft lo adoptó en la versión 4 de su navegador) vamos a poder manipular las imágenes del documento, pudiendo conseguir efectos como el conocido *rollover*, o cambio de imágenes, por ejemplo, al pasar el ratón sobre la imagen.

11.4.1 Propiedades del objeto image

border Contiene el valor del parámetro 'border' de la imagen

complete Es un valor booleano que nos dice si la imagen se ha descargado completamente o no

height Contiene el valor del parámetro 'height' de la imagen

hspace Contiene el valor del parámetro 'hspace' de la imagen

lowsrc Contiene el valor del parámetro 'lowsrc' de la imagen

name Contiene el valor del parámetro 'name' de la imagen

prototype Nos permite crear nuevos parámetros para este objeto

src Contiene el valor del parámetro 'src' de la imagen

vspace Contiene el valor del parámetro 'vspace' de la imagen

width Contiene el valor del parámetro 'width' de la imagen

En el próximo capítulo terminamos el bloque dedicado a los objetos del navegador, tratando los objetos contenidos en un formulario.

12 Los objetos del navegador. Formularios

En este capítulo finalizamos el estudio de los objetos del navegador viendo cómo manipular formularios. Este punto es especialmente importante: si aprendemos correctamente a manipular todos los objetos de un formulario, podremos hacer funciones que nos permitan validarlo antes de enviar estos datos a un servidor, ahorrándole la faena de tener que verificar la corrección de los datos enviados.

12.1 El objeto form

Este objeto es el contenedor de todos los elementos del formulario. Como ya vimos al tratar el objeto document, los formularios se agrupan en un array dentro de document. Cada elemento de este array es un objeto de tipo form.

12.1.1 Propiedades del objeto form

action Es una cadena que contiene la URL del parámetro ACTION del form, es decir, la dirección en la que los datos del formulario serán procesados.

elements Es un array que contiene todos los elementos del formulario, en el mismo orden en el que se definen en el documento HTML. Por ejemplo, si en el formulario hemos puesto, en este orden, una caja de texto, un checkbox y una lista de selección, la caja de texto será elements[0], el checkbox será elements[1] y la lista de selección será elements[2].

encoding Es una cadena que tiene la codificación mime especificada en el parámetro ENCTYPE del form.

method Es una cadena que tiene el nombre del método con el que se va a recibir/procesar la información del formulario (GET/POST)

12.1.2 Métodos del objeto form

reset(); Resetea el formulario: tiene el mismo efecto que si pulsáramos un botón de tipo RESET dispuesto en el form

submit(); Envía el formulario: tiene el mismo efecto que si pulsáramos un botón de tipo SUBMIT dispuesto en el form

Vistas ahora las propiedades y métodos del objeto form, pasamos a estudiar, uno por uno, todos los objetos contenidos en el formulario.

12.2 Los objetos text, textarea y password

Estos objetos representan los campos de texto y las áreas de texto dentro de un formulario. Además, el objeto password es exactamente igual que el text salvo en que no muestra los caracteres introducidos por el usuario, poniendo asteriscos (*) en su lugar. Los tres tienen las mismas propiedades y métodos, por ello los vemos juntos.

12.2.1 Propiedades de los objetos text, textarea y password

dafaultValue Es una cadena que contiene el valor por defecto que se le ha dado a uno de estos objetos por defecto.

name Es una cadena que contiene el valor del parámetro NAME.

value Es una cadena que contiene el valor del parámetro VALUE.

12.2.2 Métodos de los objetos text, textarea y password

blur(); Pierde el foco del ratón sobre el objeto especificado

focus(); Obtiene el foco del ratón sobre el objeto especificado

select(); Selecciona el texto dentro del objeto dado

12.3 Los objetos button

Tenemos tres tipos de botones: un botón genérico, 'button', que no tiene acción asignada, y dos botones específicos, 'submit' y 'reset'. Estos dos últimos sí que tienen una acción asignada al ser pulsados: el primero envía el formulario y el segundo limpia los valores del formulario.

12.3.1 Propiedades de los objetos button

name Es una cadena que contiene el valor del parámetro NAME.

value Es una cadena que contiene el valor del parámetro VALUE.

12.3.2 Métodos de los objetos button

click(); Realiza la acción de pulsado del botón

12.4 El objeto checkbox

Las "checkboxes" nos permiten seleccionar varias opciones marcando el cuadrito que aparece a su izquierda. El cuadrito pulsado equivale a un "sí" y sin pulsar a un "no" o, lo que es lo mismo, a "true" o "false".

12.4.1 Propiedades del objeto checkbox

checked Valor booleano que nos dice si el checkbox está pulsado o no

defaultChecked Valor booleano que nos dice si el checkbox debe estar seleccionado por defecto o no

name Es una cadena que contiene el valor del parámetro NAME.

value Es una cadena que contiene el valor del parámetro VALUE.

12.4.2 Métodos del objeto checkbox

click(); Realiza la acción de pulsado del botón

12.5 El objeto radio

Al contrario que con los checkbox, que nos permiten elegir varias posibilidades entre las dadas, los objetos radio sólo nos permiten elegir una de entre todas las que hay. Están pensados para posibilidades mútuamente excluyentes (no se puede ser a la vez mayor de 18 años y menor de 18 años, no se puede estar a la vez soltero y casado, etc.).

12.5.1 Propiedades del objeto radio

checked Valor booleano que nos dice si el radio está seleccionado o no

defaultChecked Valor booleano que nos dice si el radio debe estar seleccionado por defecto o no

length Valor numérico que nos dice el número de opciones dentro de un grupo de elementos radio

name Es una cadena que contiene el valor del parámetro NAME.

value Es una cadena que contiene el valor del parámetro VALUE.

Hay que recordar que para agrupar elementos de tipo radio, todos ellos deben tener el mismo valor en su parámetro NAME.

12.5.2 Métodos del objeto radio

click(); Realiza la acción de pulsado del botón

12.6 El objeto select

Este objeto representa una lista de opciones dentro de un formulario. Puede tratarse de una lista desplegable de la que podremos escoger alguna (o algunas) de sus opciones.

12.6.1 Propiedades del objeto select

length Valor numérico que nos indica cuántas opciones tiene la lista

name Es una cadena que contiene el valor del parámetro NAME

options Se trata de un array que contiene cada una de las opciones de la lista. Este array tiene, a su vez, las siguientes propiedades:

- defaultSelected: Valor booleano que nos indica si la opción está seleccionada por defecto
- index: Valor numérico que nos da la posición de la opción dentro de la lista
- length: Valor numérico que nos dice cuántas opciones tiene la lista
- options: Cadena con todo el código HTML de la lista
- selected: Valor booleano que nos dice si la opción está actualmente seleccionada o no
- text: Cadena con el texto mostrado en la lista de una opción concreta
- value: Es una cadena que contiene el valor del parámetro VALUE de la opción concreta de la lista

selectedIndex Valor numérico que nos dice cuál de todas las opciones disponibles está actualmente seleccionada.

12.7 El objeto hidden

Gracias a este objeto podemos almacenar información extra en el formulario de forma completamente transparente para el usuario, pues no se verá en ningún momento que tenemos estos campos en el documento. Es parecido a un campo de texto (objeto text) salvo que no tiene valor por defecto (no tiene sentido pues el usuario no va a modificarlo) y que no se puede editar.

12.7.1 Propiedades del objeto hidden

name Es una cadena que contiene el valor del parámetro NAME.

value Es una cadena que contiene el valor del parámetro VALUE.

Ahora sí, ya hemos terminado la parte de los objetos del navegador. Concluiremos el curso con los capítulos del siguiente bloque, dedicados al modelo de eventos de JavaScript. Y es que todo lo que hemos ido describiendo en estos capítulos no tendría mayor utilidad si no hubiera unos eventos que nos dice **cuándo** nos interesa realizar las acciones que ya sabemos programar.

13 Eventos en JavaScript

En sus primeras versiones, JavaScript fue dotado con un conjunto mínimo de eventos para poder dar interactividad a las páginas web. A partir de la versión 1.2 cambió su modelo de objetos para dar lugar a uno más sofisticado. Sin embargo, en este capítulo vamos a ver ese conjunto básico de eventos de los que fue dotado, dejando los cambios de la versión 1.2 para el capítulo siguiente.

Básicamente, la idea es esta: hay unas cuantas etiquetas HTML susceptibles de generar eventos. Por ejemplo, si tenemos un enlace, podemos pasar sobre él, o podemos hacer click con el ratón sobre él. Lo que se hizo fue dar nombres a estos posibles eventos, asociarlos a las etiquetas que los pueden generar, y permitir que las funciones JavaScript puedan ser llamadas al generarse dichos eventos.

Si volvemos al ejemplo del enlace: podemos hacer, de alguna manera, que al pasar sobre él, se ejecute una cierta función que tengamos definida. Esto ya fue discutido en el capítulo 2(i) del curso, por si alguien quiere revisarlo. Todo se basa en que las etiquetas HTML pueden tener como parámetro, además de los que ya tienen, los eventos que son capaces de generar, y en ese parámetro especificar la función JavaScript a ejecutar.

Siguiendo con este ejemplo: el evento de "pasar por encima de" es "onMouseOver". Supongamos que tenemos definida una función que se llame "Hacer_Algo()", la manera de que se ejecute al pasar sobre el enlace es:

```
<A HREF=''mi_enlace.html'' onMouseOver=''Hacer_Algo();''>Sucederá algo si
se pasa por aquí encima</A>
```

Visto este ejemplo, lo más importante es saber de qué eventos disponemos, y qué etiquetas los pueden generar. Esta información se resume en la tabla de la página siguiente.

Evento	Causa del evento	Directivas asociadas
onLoad	El documento se carga	<BODY>
onUnload	El documento se descarga	<BODY>
onMouseOver	El ratón pasa sobre una zona	<A HREF>, <AREA>
onMouseOut	El ratón sale de una zona	<A HREF>, <AREA>
onSubmit	Se envía un formulario	<FORM>
onClick	Se pulsa el ratón sobre algo	<INPUT TYPE="button, submit, reset, checkbox, radio">, <AREA>, <A HREF>
onBlur	Se pierde el cursor	<INPUT TYPE="text">,<TEXTAREA>
onChange	Cambia el contenido o se pierde el cursor	<INPUT TYPE="text">,<TEXTAREA>
onFocus	Se recibe el cursor	<INPUT TYPE="text">,<TEXTAREA>
onSelect	Se selecciona un texto	<INPUT TYPE="text">,<TEXTAREA>

Junto a cada directiva podemos poner respuesta a varios eventos, siempre y cuando estos pertenezcan a los que le están asociados.