Curso de JavaScript

Versión PDF del curso existente en la web: http://rinconprog.metropoliglobal.com

Autora: Lola Cárdenas Luque
e-mail: maddyparadox@wanadoo.es

Última revisión: 6 de enero de 2002

Presentación

Este curso no es ni más ni menos que un "remake" del curso de JavaScript existente en el sitio web El Rincón del Programador. He intentado ampliar en la medida de lo posible los puntos que se me quedaron cojos, sobre todo con ejemplos, que es lo que muchos de vosotros me habeis pedido. También he aprovechado para poner en práctica lo aprendido con hojas de estilo, e intentar llevar a cabo una presentación más agradable. Los capítulos han sido segmentados para que no ocupen demasiado por fichero html, agrupándose bajo la temática común en la que estaban en la web.

Aquí no vais a encontrar menos material que en el curso inicial, al contrario: lo he revisado para ampliarlo y, si acaso, he reescrito algún párrafo. Así que si teníais suficiente con aquel curso, aquí únicamente veréis más ejemplos.

Este programa y esta versión especial del curso existen porque la web cumple un año desde que vio la luz en Metropoli2000. Desde entonces he recibido muchas visitas, muchos mensajes con dudas (que he intentado responder en la medida de mis posibilidades y mi tiempo), con ánimos, con sugerencias, y como aprecio mucho la respuesta obtenida, quiero agradeceros a todos vosotros que habeis entrado en mi página y aprendido con mis cursos, con una versión muy especial del que ha sido el curso estrella de la web durante este año: el curso de JavaScript.

No esperaba ni la acogida del curso ni la buena impresión general causada: ya había muchos cursos de JavaScript en la red, así que el hecho de que el mío sea uno de los favoritos me llena de orgullo y de ganas de seguir trabajando, pues saber que he conseguido que haya personas que han aprendido JavaScript con mi curso hace que valga la pena el esfuerzo :-)

El Rincón del Programador cumple un año, y en este año he intentado dar a la web cursos que la mantengan viva por su interés. También he cambiado en parte la estética, para ayudar a mantener una organización más coherente, para vosotros y para mí. En el próximo año me gustaría seguir haciendo muchas más cosas: ampliar los cursos, ofrecer nuevos servicios y crear nuevas secciones. Si todo marcha bien, habrá un segundo cumpleaños con más regalos y sorpresas porque, de verdad, vuestro apoyo es para mí la única motivación para seguir adelante con este sitio que pretendo sea, en muchos puntos, vuestro lugar de referencia de programación.

La ocasión merecía un regalo especial, y por ello no me he limitado a un simple fichero PDF. Aparte del programa para navegar por el curso, al que quizá encontréis otras aplicaciones, ofrezco todo el fuente del curso: HTML + CSS + JavaScript, para que podais ver (y estudiar) con tranquilidad cómo he hecho las cosas.

Me dejo de presentaciones: me enrollo más que una persiana y no quiero entreteneros más leyendo una presentación cuando de lo que tendreis ganas es de empezar el curso, así que aquí acaba la parrafada O;-)

Con cariño, dedico el primer cumpleaños de la web a quienes han colaborado conmigo de una u otra manera y, en especial, a todos mis visitantes :-)

También se lo dedico a mis amigos: María, Juanjo, Salva, Chelo (cómo me han sufrido), a dos de mis compañeros de trabajo: Ernesto, Paco (pero qué bueno es reír), a mi hermano Miguel Ángel y, por supuesto, a Pedro.

Muchas gracias por animarme a seguir :-)

Valencia, 7 de noviembre de 2001.
Lola Cárdenas Luque, webmistress de El Rincón del Programador.

Qué necesito saber

Para poder seguir este curso será imprescindible tener conocimientos de los siguientes puntos:

1. HTML: es el lenguaje de marcas utilizado para crear documentos que se publicarán en la red. Si no se conoce este lenguaje, no tiene sentido alguno aprender JavaScript.
2. Nociones de programación: es necesario tener unas nociones mínimas de programación para poder seguir este curso. La idea de este curso no es enseñar a programar, es, partiendo de una base, enseñar el lenguaje JavaScript. Por ello, aunque se mencionen los conceptos para mostrar su sintaxis en JavaScript, no se entrará en explicaciones detalladas de los mismos, pues se suponen sabidos, y si no es así, existen cursos en la red en los que se enseña estas nociones mínimas.
3. Qué herramienta usar para programar: basta con un editor de texto. Como aprenderás en el curso, JavaScript va incrustado en el propio código HTML, así que con un editor de texto y un navegador que soporte JavaScript, no tendrás problemas para poder desarrollar todo el código que se te ocurra.
4. Si se quiere conseguir resultados más llamativos, es imprescindible tener conocimientos sobre hojas de estilo así como de HTML dinámico

Hay que hacer notar una cosa: en este curso no se describen los objetos específicos que implementa el navegador Internet Explorer. Para ampliar información sobre ese tema habrá que acudir a la referencia técnica citada en la sección de enlaces, o bien consultar alguno de los cursos de HTML Dinámico (DHTML) existentes en la red.

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 usuario, 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 JavaScript).

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.

Sintaxis (I): Inclusión de código y generalidades

Todos los lenguajes de programación tienen una serie de normas de escritura (su sintaxis), y JavaScript no es una excepción. Por ello, veremos en dos bloques cómo tenemos qué escribir código JavaScript. Sintácticamente, es un lenguaje similar al C, así que si tienes algún conocimiento de C, este apartado te será sencillo.

Cómo incluir código. Generalidades

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,...).

Usando <SCRIPT>

El primer método consiste en usar la directiva pareada <SCRIPT>. Esta directiva admite dos parámetros: LANGUAGE y SRC. El primero nos dice qué lenguaje de script es el que va a ser programado en el bloque. Dada la temática del curso, parece claro que en este caso tomará el valor JavaScript, pero no es el único, ya que existen otros lenguajes de script, como VBScript. Además, podremos especificar la versión del lenguaje (1, 1.1, 1.2, 1.3). El segundo parámetro, SRC, nos permite incluir un fichero externo con el código JavaScript para que sea usado en el documento. Hay que tener en cuenta que este parámetro puede usarse a partir de la versión 1.2.

Debemos tener en cuenta 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">
 <!--
    Código 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.

Vamos a ver nuestro primer ejemplo:

```
<HTML>
<HEAD></HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
<!--
  document.write("Hola mundo! :-)");
//-->
</SCRIPT>
</BODY>
</HTML>
```

En el apartado de objetos del navegador presentaremos en sociedad a la función `write` del objeto `document`, así que por ahora únicamente anticipo que sirve para escribir texto ;-)

Como respuesta a eventos

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="Código_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="GoToNoWhere.html" onMouseOver="alert('Por aquí no pasas');">Pasa
    por aquí si eres valiente</A>
```

Un consejo: no useis este ejemplo 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, a la gente suele sentarle bastante mal. Avisados quedais :)

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.

Generalidades

Vemos para finalizar el capítulo los siguientes elementos básicos que presenta el lenguaje:

- Es "Case Sensitive", es decir, distingue mayúsculas de minúsculas.
- Comentarios: `/* ... */` para encerrar un bloque y `//` para comentarios de una línea.
- Cada sentencia ha de terminar en `;`
- Encerrando código entre llaves `{ ... }` lo agrupamos. Se verá su sentido cuando tratemos las estructuras de control.

Sintaxis (I): Variables y constantes

Ya hemos visto cómo incluir código JavaScript en los documentos HTML, así que pasamos ahora a ver qué reglas hemos de seguir para escribir este código. En este capítulo nos ocupamos de las variables y las constantes.

Variables

JavaScript tiene la peculiaridad de ser un lenguaje ***débilmente tipado***, esto es, se 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";
```

Ámbito 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>`

Constantes

Las constantes no existen como tales en JavaScript. Salvando el caso del objeto `Math`, que veremos más adelante, no las tenemos disponibles. Si se quiere usar una constante en el programa, no quedará más remedio que declarar una variable, asignarle un valor desde el principio del programa, y procurar no tocarlo. Es lo más parecido a las constantes que se puede obtener.

Sintaxis (I): Tipos de datos

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

El tipo entero y el tipo flotante se engloban dentro del tipo `Number`. Los `nulos` y los `indefinidos` no son realmente tipos de datos: son valores especiales. Es más, para JavaScript, el tipo nulo (`null`) es un objeto.

Enteros

Podemos representar números enteros de tres formas distintas: en base 10 (la usual), en base 16 (hexadecimal) o en base 8 (octal). 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** (cero). Por ejemplo, `01763` se refiere a `1763` en octal.

Flotantes

Con ellos podremos representar números en coma flotante, usando notación decimal o científica. La notación científica consiste en lo siguiente:

Se toma el número decimal y se normaliza, es decir, se corre el punto decimal tantos puestos a la izquierda o a la derecha como sean necesarios para que haya un único número distinto de cero antes del punto. Para ello, será necesario acompañar este desplazamiento del punto con la respectiva multiplicación o división por potencias de 10. El formato resulta:

```
X.YYYYYYeNN
```

donde e (también puede ser E) significa "exponente", y `NN` es el número al que elevar 10 para obtener el número.

Por ejemplo, el número `123.298` se escribe `1.23298E2`, mientras que `0.00123` se escribe `1.23E-3`

Booleanos

Tomarán los valores de verdad `true` o `false` (1 ó 0). El resultado de evaluar una condición será un valor booleano.

Nulos

`null` es un valor especial que nos dice que un objeto no ha sido asignado.

Indefinidos

El valor `undefined` (indefinido) nos dice que la variable a la que queremos acceder no le ha sido asignado valor alguno, y por tanto permanece indefinida.

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 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...

```
\b   Espacio hacia atrás
\f   Alimentación de línea
\n   Nueva línea
\r   Retorno de carro
\t   Tabulación
\\   Barra invertida: \
\'   Comilla simple: '
\"   Comilla doble: "
```

Objetos

Una variable puede ser un objeto, en cuyo caso corresponde a este tipo. Veremos con detalle los objetos en un tema aparte.

Funciones

Las funciones también son un tipo de dato especial: si una variable es de tipo `function`, en realidad se trata de una función.

Operadores

Los operadores son símbolos que denotan operaciones que pueden realizarse entre elementos llamados operandos. Si tenemos un operador, léase $ (por no poner uno conocido que confunda), que sabe operar con dos operandos, entonces a $ b debe entenderse como el resultado de operar a con b según lo que indique la operación $. Por ejemplo, + es un operador que admite dos operandos y tal que el resultado de hacer a + b es el resultado de sumar a y b.

El operador más básico es el operador binario de asignación, =. Si escribimos en el código a = b, estamos asignando a a el contenido de b en ese momento.

Veamos un ejemplo:

```
<HTML>
<HEAD></HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
<!--
var a = 4, b = 3; /* Variables globales */

document.write('Valor de a: ' + a + '<BR>Valor de b: '
  + b + '<P>');

a = b;
document.write('Valor de a: ' + a + '<BR>Valor de b: '
  + b);

//-->
</SCRIPT>
</BODY>
</HTML>
```

En el ejemplo podemos ver algunos de los elementos ya explicados, como el uso de comentarios, y otros no tan conocidos, como el uso del operador + de esta forma. Resulta que 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";
```

Operadores: Aritméticos

Los operadores aritméticos son binarios (necesitan dos operandos), y realizan sobre sus operandos alguna de las operaciones aritméticas conocidas. En concreto, tenemos:

```
+     Suma
-     Resta
*     Producto
/     Cociente
%     Módulo
```

En JavaScript tenemos una versión especial, heredada del lenguaje C, que junta el operador de asignación con los vistos aquí:

```
+=     b += 3  equivale a   b = b + 3
-=     b -= 3  equivale a   b = b - 3
*=     b *= 3  equivale a   b = b * 3
/=     b /= 3  equivale a   b = b / 3
%=     b %= 3  equivale a   b = b % 3
```

Además, heredados también del lenguaje C, tenemos los incrementos y decrementos: `++` 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;
```

pues 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 valdrían lo mismo.

Por último, dentro de los operadores aritméticos tenemos el operador unario `-`, que aplicado delante de una variable, le cambia el signo.

A continuación vemos algunos ejemplos de operaciones y su resultado:

```
<HTML>
<HEAD></HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
<!--
/* Partimos de dos variables, 'a' y 'b', y vamos a realizar
   distintas operaciones con ellas */
var a = 10, b = 6;

document.write('a = ' + a + ', b = ' + b + '<P>');
document.write('a + b = ' + (a + b) + '<BR>');
document.write('a - b = ' + (a - b) + '<BR>');
document.write('a * b = ' + (a * b) + '<BR>');
document.write('a / b = ' + (a / b) + '<BR>');
document.write('a % b = ' + (a % b) + '<BR>');
document.write('a++ : ' + (a++) + '<P>');

/* Notar que aquí ya se ha incrementado 'a' */
document.write('-a = ' + (-a) + '<P>');

document.write(' a += b : a = ' + (a += b) + '<BR>');
document.write(' a %= b : a = ' + (a %= b) + '<BR>');

//-->
</SCRIPT>
</BODY>
</HTML>
```

Operadores: Comparación

Los operadores de comparación son binarios y su resultado es un booleano (un valor de verdad). Nos permiten expresar si una relación de igualdad/desigualdad es cierta o falsa dados los operandos. Tenemos los siguientes:

```
==        igual
!=        distinto
>         mayor que
<         menor que
>=        mayor o igual que
<=        menor o igual que
```

A partir de JavaScript 1.3 esta lista se amplia con dos operadores más:

```
===        estrictamente igual
!==        estrictamente distinto
```

El primero devuelve `true` cuando los operadores toman el mismo valor y además son del mismo tipo. El segundo devuelve `true` cuando los operadores son distintos y de tipo distinto. Es decir, si los operandos tienen distinto valor, pero tienen el mismo tipo, **no** son estrictamente distintos.

Veamos un sencillo ejemplo para aclarar conceptos:

```
<HTML>
<HEAD></HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
<!--

/* Partimos de dos variables, 'a' y 'b', y vamos a realizar
   distintas operaciones con ellas */

var a = 10, b = 6;

document.write('a = ' + a + ', b = ' + b + '<P>');

document.write('a == b : ' + (a == b) + '<BR>');
document.write('a != b : ' + (a != b) + '<BR>');
document.write('a < b : ' + (a < b) + '<BR>');
document.write('a > b : ' + (a > b) + '<BR>');
document.write('a <= b : ' + (a <= b) + '<BR>');
document.write('a >= b : ' + (a >= b) + '<BR>');
```

```
a = '6';
```

```
document.write('Ahora a = ' + a + ' (tipo: ' + typeof a + ')<BR>');
document.write(' a === b : ' + (a === b) + '<BR>');

a = 6;
document.write('Ahora a = ' + a + ' (tipo: ' + typeof a + ')<BR>');
document.write(' a === b : ' + (a === b) + '<BR>');
document.write(' a !== b : ' + (a !== b) + '<BR>');

a = '6';
document.write('Ahora a = ' + a + ' (tipo: ' + typeof a + ')<BR>');
document.write(' a !== b : ' + (a !== b) + '<BR>');

//-->
</SCRIPT>
</BODY>
</HTML>
```

Aparece en este ejemplo otro operador, typeof, que estudiaremos en el apartado Resto de los operadores.

Operadores: Lógicos

Los operadores lógicos sirven para componer condiciones más simples por medio de las reglas de la **y**, **o** y **no** lógicas. Nos permiten expresar condiciones compuestas de las que queremos averiguar su valor de verdad.

```
&&     AND ('y' lógica)
||     OR ('o' lógica)
!      NOT ('no' lógica)
```

Por ejemplo, si tenemos tres números, podemos querer saber si el primero es a la vez mayor que el segundo y el tercero, o si es mayor que alguno de ellos, o si no es mayor que ninguno. Con el adecuado uso de paréntesis, podemos agrupar como nos convenga las condiciones.

Veamos un ejemplo:

```
<HTML>
<HEAD></HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
<!--
/* Partimos de tres variables, 'a', 'b' y 'c', y vamos a realizar
   distintas operaciones con ellas */
var a = 10, b = 6, c = 11;
document.write('a = ' + a + ', b = ' + b + ', c = ' + c + '<P>');
document.write('(a > b) && (a > c) : '
  + ((a > b) && (a > c)) + '<BR>');
document.write('(a > b) && (a < c) : '
  + ((a > b) && (a < c)) + '<BR>');
document.write('!(a < b) : ' + !(a < b) + '<BR>');
document.write('(a > b) || (a > c) : '
  + ((a > b) || (a > c)) + '<BR>');
//-->
</SCRIPT>
</BODY>
</HTML>
```

La primera comparación nos dice si 'a' es mayor que 'b' y a la vez mayor que 'c'. La segunda, si 'a' es mayor que 'b' y a la vez menor que 'c'. La tercera nos dice si no se cumple que 'a' sea menor que 'b'. La cuarta nos dice si 'a' es mayor que 'b' o bien 'a' es mayor que 'c'.

Operadores: A nivel de bit

JavaScript nos facilita los siguientes operadores para realizar operaciones a nivel de bits:

```
&        AND bit a bit
|        OR  bit a bit
^        XOR bit a bit
~        NOT bit a bit
>>       rotación a derecha
<<       rotación a izquierda
```

Resulta más intuitivo el resultado si escribimos los números en binario, así que en el siguiente ejemplo, asignamos valores a dos variables, y realizamos algunas operaciones.

```
 <HTML>
 <HEAD>
<SCRIPT LANGUAGE="JavaScript">
<!--
  function NumToBase(num, base, prefijo, sufijo)  {
    var Conversion;
    if(typeof base == 'undefined')    base = 16;
    if(typeof prefijo == 'undefined') prefijo = '0x';
    if(typeof sufijo == 'undefined')  sufijo = '';

    if (num < 0)
      Conversion = '-' + prefijo + Math.abs(num).toString(base) + sufijo;
    else
      Conversion = prefijo + num.toString(base) + sufijo;

    return(Conversion);
    }
//-->
</SCRIPT>
 </HEAD>
```

```
<BODY>
<SCRIPT LANGUAGE="JavaScript">
<!--
/* Partimos de dos variables, 'a' y 'b', y vamos a realizar
   distintas operaciones con ellas */
var a = 0x10, b = 0x14;

document.write('a = ' + NumToBase(a, 2, 'B:') + ', b = '
  + NumToBase(b, 2, 'B:') + '<P>');
```

```
document.write('a & b : ' + NumToBase(a & b, 2, 'B:') + '<BR>');
document.write('a | b : ' + NumToBase(a | b, 2, 'B:') + '<BR>');
document.write('a ^ b : ' + NumToBase(a ^ b, 2, 'B:') + '<BR>');
document.write('~a: ' + NumToBase(~a, 2, 'B:') + '<BR>');
document.write('a << 2: ' + NumToBase(a << 2, 2, 'B:') + '<BR>');
document.write('a >> 2: ' + NumToBase(a >> 2, 2, 'B:') + '<BR>');

//-->
</SCRIPT>
</BODY>
</HTML>
```

En este ejemplo hemos realizado una tarea más: hemos definido una función, `NumToBase(num, base, prefijo, sufijo)`, que realizará la conversión del número en decimal a la base que necesitemos para representarlo como una cadena que es la que vemos como resultado en el navegador. En el apartado `Sintaxis (II)` veremos cómo se definen y usan funciones en JavaScript, aunque este primer ejemplo ya nos da una idea. Utilizamos también el objeto `Math` para poder usar su método `abs`, que nos da el valor absoluto del número pasado como argumento. Este objeto se explicará en el apartado `Objetos del navegador`.

Operadores: Resto

En este punto completamos los operadores de que dispone JavaScript:

```
new, delete, void, typeof
```

Vayamos uno a uno.

`new` es un operador que sirve para crear nuevas instancias de un objeto vía su constructor. Aprenderemos a crear y usar objetos en el capítulo de objetos.

`delete` es un operador que sirve para destruir una instancia concreta de un objeto.

`void` es un operador que permite evaluar una expresión sin que devuelva valor alguno. Podría considerarse como una función especial.

`typeof` es un operador que nos devuelve una cadena que describe el tipo de dato que corresponde con el objeto (variable, función, etc) que se escribe a continuación. Pertenece al lenguaje a partir de la versión 1.1.

Vemos un sencillo ejemplo para mostrar resultados obtenidos con estos operadores:

```
<HTML> <HEAD> </HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
<!--
document.write('typeof NaN: ' + (typeof NaN) + '<BR>');
document.write('typeof true: ' + (typeof true) + '<BR>');
document.write('typeof null: ' + (typeof null) + '<BR>');
document.write('typeof 45e-3: ' + (typeof 45e-3) + '<BR>');
document.write('typeof parseInt: ' + (typeof parseInt) + '<BR>');
document.write('typeof "Hola mundo": ' + (typeof "Hola mundo") + '<BR>');
document.write('typeof Math: ' + (typeof Math) + '<P>');

var a = new Array(1,2);
document.write('a[0]: ' + a[0] + '<BR>');
delete a;
document.write('typeof a: ' + (typeof a));
//-->
</SCRIPT>
</BODY> </HTML>
```

Todos los objetos que aquí aparecen que quizá ahora mismo no estén muy claros, serán explicados con detalle a lo largo del curso (como es el caso del valor `NaN`, la función `parseInt` o los objetos `Array` y `Math`).

Sintaxis (II): Estructuras de control. Funciones

En este apartado veremos las distintas estructuras de control que nos ofrece JavaScript para diseñar los programas. Las dividiremos en los siguientes puntos:

1. Condicionales
2. Bucles
3. Funciones

que serán los que comentaremos detalladamente en los capítulos de este apartado.

Sintaxis (II): Condicionales

En este apartado veremos:

El condicional if

La estructura del condicional Si-Entonces-Si no sigue esta sintaxis en JavaScript:

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

La condición de comparación puede ser compuesta (usando para ello operadores lógicos que unirán las distintas condiciones), y podemos tener varios if anidados. Además, el bloque else no es obligatorio. Por último, cabe señalar que si el código a ejecutar es una única sentencia, entonces no es necesario encerrarla entre llaves.

Veamos un sencillo ejemplo:

```
<HTML>
<HEAD></HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
<!--
var Nombre, Edad;
Nombre = prompt('No te conozco, ¿cómo te llamas?', '');
Edad = prompt('¿Y qué edad tienes?', '');
Edad = parseInt(Edad);
if (Edad < 18)
  document.write('Vaya, ' + Nombre +', sólo tienes ' + Edad + ' años');
else
  document.write('No sabía que fueras mayor de edad, ' + Nombre);
//-->
</SCRIPT>
</BODY>
</HTML>
```

También tenemos el condicional ternario:

```
  [ Expresión ] ? [ Sentencia1 ] : [ Sentencia2 ] ;
```

que evalúa `[ Expresión ]`. El resultado es un booleano. Si es `'true'` se ejecutará `[ Sentencia1 ]` y si es `'false'`, se ejecutará `[ Sentencia2 ]`

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")'`.

El condicional switch

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

```
 switch(condicion) {
   case caso1 :
     sentencias para caso1;
     break;
        ......
   case casoN :
     sentencias para casoN;
     break;
   default :
     sentencias por defecto;
     break;
   }
```

Son útiles, por ejemplo, cuando se ofrece una serie de opciones de entre las cuales se elige una, como podemos ver en el ejemplo:

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
<!--
function Proceder() {
  var Opcion, Valor = 100;
  Opcion = prompt('Elige un número de descuento', '1');

  /* Nos aseguramos de que está en el rango correcto
     de opciones */

  parseInt(Opcion);
  if (Opcion < 1 || Opcion > 4) Opcion = 1;
```

```
   switch(Opcion) {
```

```
     case '1':
       Valor *= 0.95;
       break;
     case '2':
       Valor *= 0.9;
       break;
     case '3':
       Valor *= 0.85;
       break;
     case '4':
       Valor *= 0.8;
       break;
     default:
       Valor *= 1;
       break;
     }

  alert('Tu descuento: ' + Valor);
  }
//-->
</SCRIPT>
</HEAD>
<BODY>

Elige un descuento a realizar al valor 100: <P>
<OL>
<LI>5%
<LI>10%
<LI>15%
<LI>20%
</OL>

<FORM>
<INPUT TYPE="BUTTON" onClick="Proceder();" VALUE="Calcular">
</FORM>

</BODY>
</HTML>
```

En este ejemplo vemos más elementos nuevos que iremos explicando. Hemos vuelto a definir una función, y hemos hecho uso de la llamada a dicha función vía el evento `Click` del botón del formulario. No he querido hacerlo más complejo usando formularios de forma más intensa, por no complicar aún los conceptos. Tendremos todo un capítulo para dedicarnos a ellos. Además, este es el primer ejemplo donde definimos algo en la cabecera del documento. Hemos definido ahí la función, porque así, cuando cargue la parte del documento que la llama, ya estará disponible.

Sintaxis (II): Bucles

En este apartado estudiamos las distintas construcciones que nos ofrece JavaScript para programar bucles:

El bucle for

Este bucle presenta el mismo aspecto que en el lenguaje C:

```
for([Inicialización]; [Condición]; [Expresión de actualización])
  {
    Instrucciones a repetir
  }
```

Se ejecuta la inicialización, se evalúa la condición y se actualizan los valores dados en la expresión de actualización. Ninguna de las tres expresiones son obligatorias, es decir, perfectamente podríamos escribir código como `for(;;);` (sabiendo que esto es un bucle infinito, claro).

Para ilustrar el uso de `for`, recurriremos al ejemplo típico de crear las tablas de multiplicar:

```
<HTML> <HEAD></HEAD> <BODY>
<SCRIPT LANGUAGE="JavaScript"> <!--
  var i, j;  // Índices para recorrer la tabla
  var MaxNumeros = 10;
  document.write("<TABLE BORDER='0'>");
  for(i = 1; i <= MaxNumeros; i++) {
    document.write("<TR><TH>Tabla de multiplicar del "
      + i + "</TH></TR>");
    for(j= 1; j <= 10; j++)
      document.write("<TR><TD>" + i + ' x ' + j + ' = '
        + (i * j) + "</TD></TR>");
    }
  document.write("</TABLE>");
//--> </SCRIPT>
</BODY> </HTML>
```

El bucle while

Este bucle también sigue la sintaxis del while del lenguaje C:

```
while(Condición) {
 Instrucciones a repetir
 }
```

Por ejemplo:

```
<HTML>
<HEAD></HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
<!--
 var v = new Array('1', '8', '9', '3', '5', '4', '6', '32');
 var Numero, i, Enc;
 Numero = prompt("Introduce un número:", "");
 Enc = false;
 i = 0;
 while ( (i < v.length) && !Enc )  {
   if (v[i] == Numero) Enc = true;
   i++;
   }
 if (Enc)
   alert('El número ' + Numero + ' está en el vector\n'
     + 'en la posición ' + (i - 1));
 else alert('El número ' + Numero + ' no está en el vector');
//-->
</SCRIPT>

</BODY>
</HTML>
```

En este ejemplo hemos realizado una búsqueda dentro de un array (estudiaremos este objeto más adelante) siguiendo un esquema básico de búsqueda. Salvo el objeto Array, los demás elementos son ya conocidos por lo que no es complicado entender el ejemplo.

El bucle do ... while

También sigue la misma sintaxis que en C (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: el primero salta fuera del bucle más interno, y el segundo salta las instrucciones que quedaran por ejecutar y vuelve a la cabecera del bucle.

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

El bucle for ... in

En JavaScript también disponemos de:

```
for(var 'Propiedad' in 'Objeto') {
  Instrucciones a repetir
  }
```

que produce una iteración a través de todas las propiedades de un objeto (a su tiempo veremos el punto de objetos). Por ejemplo, este código recorre a `document` y nos muestra todas sus propiedades.

```
<HTML>
<HEAD></HEAD>
<BODY>

<SCRIPT LANGUAGE="JavaScript">
<!--

 for(var i in document)
   document.write('document.' + i + ' = ' + document[i] + '<BR>');

//-->
</SCRIPT>

</BODY>
</HTML>
```

La construcción with

Esta construcción, cuya sintaxis es:

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

nos ahorra escribir siempre el nombre de un objeto si vamos a utilizarlo muchas veces para acceder a sus métodos, propiedades,...

Por ejemplo:

```
<HTML>
<HEAD></HEAD>
<BODY>

<SCRIPT LANGUAGE="JavaScript">
<!--

 with(document) {
   write('Usando el with para abreviar escribir document.write ');
   write('cuando usas mucho este método, y muy seguido. Este ');
   write('sólo es una prueba.');
   }

//-->
</SCRIPT>

</BODY>
</HTML>
```

Sintaxis (II): 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 NombreFuncion(arg1, ..., argN)  {
  Código de la función

  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="NombreFuncion(arg1,...);">Pulse
por aquí, 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 `NombreFuncion.arguments`), y podemos saber cuántos argumentos se han entrado viendo el valor de `NombreFuncion.arguments.length`

Por ejemplo:

```
function Cuenta()  {
  document.write("Me han llamado con " + Cuenta.arguments.length
      + " argumentos<P>");


  for(i = 0; i < Cuenta.arguments.length; i++)
    document.write("Argumento " + i + ": "
      + Cuenta.arguments[i] + "<BR>");
 }
```

La podemos llamar con:

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

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

Con detalle, el código sería este:

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
<!--

 function Cuenta()  {
   document.write("Me han llamado con " + Cuenta.arguments.length
       + " argumentos<P>");

   for(i = 0; i < Cuenta.arguments.length; i++)
     document.write("Argumento " + i + ": "
       + Cuenta.arguments[i] + "<BR>");
   }

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

<SCRIPT LANGUAGE="JavaScript">
<!--

 Cuenta("Hola","a","todos");

//-->
</SCRIPT>

</BODY>
</HTML>
```

Ahora ya sabemos cómo crear nuestras propias funciones :-)

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 sobre las 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...

Las seis funciones que trataremos serán estas:

1. parseInt
2. parseFloat
3. escape
4. unescape
5. isNaN
6. eval

Funciones: parseInt y parseFloat

Las funciones tratadas en este capítulo nos serán de utilidad para convertir cadenas en valores numéricos: enteros y flotantes.

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).

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`.

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
```

Funciones: escape, unescape

Estas funciones nos permiten tratar con cadenas que contienen códigos de escape:

1. escape
2. unescape

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): { / . - _ * }

Ejemplos:

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

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

unescape(códigos);

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 '`, ...

Funciones: isNan, eval

Terminamos el bloque de funciones propias del lenguaje con estas dos funciones, que nos dicen si un argumento es numérico y nos evalúan la expresión que le introduzcamos, respectivamente.

1. isNaN
2. eval

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(expresión);

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`.

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. Como los objetos no suelen ser tratados en primeros cursos de programación, se intentará dar una primera idea de lo que son, para luego detallar cómo son en JavaScript.

Objetos: Introducción

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 Ventana1, Ventana2
```

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:

```
Ventana1.x0 = 3
Ventana2.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 `Ventana1.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.

Objetos: Cómo son en JavaScript

JavaScript 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 sencillo 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 creación 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 pasará un parámentro
    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 área es: " + area);
   }

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

```
  /* Esta función sirve para probar la creación 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";   // Añadimos 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.

Los objetos del lenguaje

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.

Objetos del lenguaje: 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.

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.

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("Tamaño 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 pequeño: " + 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>
```

Objetos del lenguaje: 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 usar 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.

Propiedades del objeto Array

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

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", "Cariño");
  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>
```

Objetos del lenguaje: 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.

Propiedades del objeto Math

`E`	Número '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`	Número PI.
`SQRT1_2`	Raíz cuadrada de 1/2.
`SQRT2`	Raíz cuadrada de 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 número 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 número.
`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)`

Objetos del lenguaje: 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.

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.

Objetos del lenguaje: 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.

Objetos del lenguaje: 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.

Objetos del lenguaje: Function

Ya hablamos de este objeto, sin mencionar que era un objeto, cuando estudiamos la 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.

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.

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 document), `'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.

Objetos del navegador: 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 objeto `window` es el 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.

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.

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, características)`
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 dados 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 ejemplo 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>
```

Objetos del navegador: 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.

Dado que `frame` coincide en sus propiedades y métodos con `window`, simplemente daremos un rápido listado de repaso a los mismos, remitiendo pues la explicación de ellos a los vistos para `window`.

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`

Métodos del objeto frame

```
alert(mensaje)
blur()
clearInterval(id)
clearTimeout(nombre)
close()
confirm(mensaje)
focus()
moveBy(x, y)
moveTo(x, y)
open(URL, nombre, características)
```

Características:

```
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)
```

Vamos a ver ahora un ejemplo. Tendremos tres frames dispuestos de la siguiente forma: dos filas que dividen la pantalla en superior e inferior y, dentro de la inferior, dos columnas que dividen la pantalla en izquierda y derecha. Pondremos unas opciones en el frame superior cuyo efecto se verá en los inferiores.

El documento de definición de los frames será:

```
<HTML> <HEAD>  <TITLE> Ejemplo frames </TITLE> </HEAD>
<FRAMESET ROWS="50%,*">
 <FRAME SRC="ej1JS8_sup.html" NAME="Superior" SCROLLING="NO">
 <FRAMESET COLS="50%,*">
  <FRAME SRC="ej1JS8_izq.html" NAME="Izquierdo">
  <FRAME SRC="ej1JS8_der.html" NAME="Derecho">
 </FRAMESET>
</FRAMESET>
<!-- Contenido para navegadores sin frames -->
<NOFRAMES>  <BODY BGCOLOR=white>
 Para poder ver este ejemplo es necesario que el navegador
 soporte frames, lo siento.
 </BODY> </NOFRAMES>
</HTML>
```

El frame superior vendrá dado por:

```
<HTML> <HEAD>  <TITLE> Frame superior </TITLE>
<SCRIPT LANGUAGE="JavaScript"> <!--
 function EscribirEnFrame(Frame, Texto) {
   var QueFrame;

   QueFrame = Frame[0].checked ? top.Izquierdo : top.Derecho;
   QueFrame.document.write(Texto);
   }
//--> </SCRIPT> </HEAD>

<BODY>
Escribe un texto, elige el frame, y pulsa el botón para que se escriba
el texto en el frame: <P>
<CENTER> <FORM>
Texto: <INPUT TYPE=TEXT SIZE="20" MAXLENGTH="50" NAME="Palabras"><BR>
Frame: <INPUT TYPE=RADIO NAME="FrameElegido" VALUE="Izquierdo" CHECKED>Izquierdo
       <INPUT TYPE=RADIO NAME="FrameElegido" VALUE="Derecho">Derecho<BR>
<INPUT TYPE=BUTTON VALUE="Escribir"
  onClick="EscribirEnFrame(this.form.FrameElegido, this.form.Palabras.value);">
</FORM> </CENTER>
</BODY> </HTML>
```

El frame izquierdo será:

```
<HTML> <HEAD>
 <TITLE> Frame izquierdo </TITLE>
</HEAD>

<BODY> </BODY> </HTML>
```

Y finalmente el derecho:

```
<HTML> <HEAD> <TITLE> Frame derecho </TITLE> </HEAD>
<BODY> </BODY> </HTML>
```

Los documentos para los frames izquierdo y derecho están vacíos puesto que en principio, para este ejemplo no es necesario escribir texto alguno. Todo el tratamiento hecho para el formulario se verá claro en el capítulo en que se explican los componentes de este elemento.

Objetos del navegador: 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
```

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.

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.

Veamos un sencillo ejemplo de uso:

```
<HTML> <HEAD></HEAD> <BODY>
Déjame que te diga algunas cosas sobre tu <TT>location</TT>: <P>
<SCRIPT LANGUAGE="JavaScript"> <!--
  var i;

  for(i in location)
    document.write('location.' + i + ' = ' + location[i] + '<BR>');
//--> </SCRIPT>
</BODY> </HTML>
```

Objetos del navegador: 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.

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.

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(posición)`
: 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.

Objetos del navegador: navigator

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

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` (NS), `systemLanguage` (IE)
: Cadena de dos caracteres (IE) o cinco (NS) 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.

Métodos del objeto navigator

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

En el siguiente ejemplo, conseguimos que este objeto nos cuente algunas de sus cosas :-):

```
<HTML> <HEAD></HEAD> <BODY>
Déjame que te diga algunas cosas sobre tu <TT>navigator</TT>: <P>

<SCRIPT LANGUAGE="JavaScript"> <!--
var Explorer = navigator.language ? false : true;
with(document) {
  write('Por una parte, navigator.appCodeName = ' + navigator.appCodeName + '<BR>');
  write('Por otro lado, navigator.appName = ' + navigator.appName + '<BR>');
  write('Además, navigator.appVersion = ' + navigator.appVersion + '<BR>');

  if (Explorer)
    write('Por si fuera poco, navigator.systemLanguage = '
      + navigator.systemLanguage + '<BR>');
  else
    write('Por si fuera poco, navigator.language = '
      + navigator.language + '<BR>');
```

```
    write('Y ahí no termina, pues navigator.mimeTypes = '
      + navigator.mimeTypes + '<BR>');
    write('Pero es que también navigator.platform = '
      + navigator.platform + '<BR>');
    write('No olvidemos, por otro lado, que navigator.plugins = '
      + navigator.plugins + '<BR>');
    write('Finalmente, pero no menos importante, navigator.userAgent = '
      + navigator.userAgent);
    }
  //--> </SCRIPT> </BODY> </HTML>
```

Objetos del navegador: 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.

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

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

Al fin vemos el `document.write` que nos ha venido acompañando a lo largo del curso :-)

Desarrollamos aquí un pequeño ejemplo que toca algunas de las propiedades de `document`:

```
<HTML> <HEAD>
<SCRIPT LANGUAGE="JavaScript"> <!--
function CambiarColorFondoDoc(RadioBtn) {
  var i = 0, Enc = false;
  while ( (i < RadioBtn.length) && !Enc ) {
    if (RadioBtn[i].checked) Enc = true;
    i++;
    }
  document.bgColor = RadioBtn[i-1].value;
  }

function VerFechaUltimaModificacion() { alert(document.lastModified); }
function VerTitulo() { alert(document.title); }
//--> </SCRIPT> </HEAD>

<BODY>
Quiero: <P>
<OL>
<LI>Ver la <A HREF="javascript:VerFechaUltimaModificacion();">fecha de
  la última modificación</A> hecha al documento.
<LI>Cambiar el color de fondo: <P>
<FORM>
 <INPUT TYPE=RADIO NAME="Color" VALUE="silver" CHECKED>Plata<BR>
 <INPUT TYPE=RADIO NAME="Color" VALUE="red">Rojo<BR>
 <INPUT TYPE=RADIO NAME="Color" VALUE="lime">Lima<BR>
 <INPUT TYPE=RADIO NAME="Color" VALUE="navy">Azul marino<BR>
 <INPUT TYPE=RADIO NAME="Color" VALUE="white">Blanco<BR>
 <INPUT TYPE=RADIO NAME="Color" VALUE="orange">Naranja<BR>
 <INPUT TYPE=RADIO NAME="Color" VALUE="yellow">Amarillo<BR>
 <INPUT TYPE=BUTTON VALUE="Cambiar Color"
   onClick="CambiarColorFondoDoc(this.form.Color);">
</FORM>
<LI>Ver el <A HREF="javascript:VerTitulo();">título del documento</A>
</OL>

</BODY> </HTML>
```

Objetos del navegador: link

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

Propiedades del objeto link

`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

Veamos un ejemplo:

```
<HTML> <HEAD> </HEAD>
<BODY>
Recorramos todos los enlaces que vemos a continuación, y a ver que nos dice
JavaScript de sus propiedades: <P>
<A HREF="http://www.notoy.es/EnlaceFalso1.html">Enlace falso número 1</A><BR>
<A HREF="http://www.notoy.es/EnlaceFalso2.html">Enlace falso número 2</A><BR>
<A HREF="#OtraSeccion">Enlace a una sección</A><P>

<SCRIPT LANGUAGE="JavaScript"> <!--
var i;
for(i = 0; i < document.links.length; i++)
  with(document) {
    write('link[' + i + '].hash = ' + links[i].hash + '<BR>');
    write('link[' + i + '].href = ' + links[i].href + '<BR>');
    write('link[' + i + '].host = ' + links[i].host + '<BR>');
    write('link[' + i + '].port = ' + links[i].port + '<BR>');
    write('link[' + i + '].protocol = ' + links[i].protocol + '<BR>');
    write('link[' + i + '].target = ' + links[i].target + '<P>');
    }
//--> </SCRIPT>
</BODY> </HTML>
```

Objetos del navegador: anchor

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

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`

Veamos un sencillo ejemplo de uso:

```
<HTML> <HEAD> </HEAD>
<BODY>
Recorramos todos los enlaces que vemos a continuación, y a ver que nos dice
JavaScript de sus propiedades: <P>

<A HREF="EnlaceFalsoM.html">Enlace falso número M</A><BR>
<A NAME="EnlaceFalso1" TARGET="A">Enlace falso número 1</A><BR>
<A NAME="EnlaceFalso2">Enlace falso número 2</A><P>

<SCRIPT LANGUAGE="JavaScript"> <!--
var i;

for(i = 0; i < document.anchors.length; i++)
  with(document) {
    write('anchors[' + i + '].name = ' + anchors[i].name + '<BR>');
    write('anchors[' + i + '].target = ' + anchors[i].target + '<P>');
    }
//--> </SCRIPT>

</BODY> </HTML>
```

Objetos del navegador: 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.

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

Vamos a ver un ejemplo en el que cambiaremos el tamaño de la imagen según decidamos:

```
<HTML> <HEAD>
<SCRIPT LANGUAGE="JavaScript"> <!--

function CambiarTamImagen(QueImagen, QueOpcion, QueEscala) {
  var i = 0, Enc = false, Escala;

  while( (i < QueOpcion.length) && !Enc )
    if (QueOpcion[i++].checked) Enc = true;

  Escala = parseFloat(QueEscala.value);

  if (isNaN(Escala) || (!isNaN(Escala) && (Escala < 0)) )
    Escala = 1;

  QueImagen.width *= 1;
  QueImagen.height *= 1;
```

```
  switch(i-1) {
    case 0:
      QueImagen.width *= Escala;
      break;
    case 1:
      QueImagen.height *= Escala;
      break;
    case 2:
      QueImagen.width *= Escala;
      QueImagen.height *= Escala;
      break;
    }
  }

//--> </SCRIPT> </HEAD>
<BODY>
Aquí tienes algunas opciones para trastear con la imagen: <P>

<FORM>
<INPUT TYPE=RADIO NAME="Escalar" VALUE="EnX" CHECKED>
Escalar la imagen en X. <BR>
<INPUT TYPE=RADIO NAME="Escalar" VALUE="EnY">
Escalar la imagen en Y. <BR>
<INPUT TYPE=RADIO NAME="Escalar" VALUE="EnAmbos">
Escalar la imagen (ambos ejes).<P>

Factor (mayor o igual que cero):
<INPUT TYPE=TEXT SIZE=10 MAXLENGTH=15 NAME="Escala"> <P>

<INPUT TYPE=BUTTON VALUE="Aplicar cambios"
 onClick="CambiarTamImagen(this.form.Imagen,
   this.form.Escalar, this.form.Escala);"> <P>

<CENTER><IMG SRC="GatitoLapiz.gif" NAME="Imagen"></CENTER>
</FORM>
</BODY> </HTML>
```

La escala a aplicar debe ser un número válido mayor o igual que cero, que será el factor por el que se multiplique el tamaño en X o en Y (o ambos) de la imagen. Si poneis como valor de escala el cero, al multiplicarse por cero, a partir de ese momento ya no vereis la imagen. Podeis poner números con decimales. El efecto de la escala siempre será: si la escala es igual a 1, la imagen se queda como está, si es mayor que 1 la imagen se hace más grande, y si es menor que 1, la imagen se hace más pequeña.

En cuanto al trozo de código:

```
   QueImagen.width *= 1;
   QueImagen.height *= 1;
```

hay que decir que cuando hacemos el **primer** cambio de escala, el navegador entiende que queremos aplicarlo tanto al ancho como al alto, y a partir de ahí individualmente. Para evitar que la primera vez que se cambie de escala se cambie en ambas direcciones, se ejecutan esas líneas previamente, con lo que ya hay un cambio de escala (aunque sea dejarlas igual) y ya podemos cambiar la escala en el eje que queramos.

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. Los formularios pueden tener otras muchas utilidades, y será conveniente pues saber manipularlos bien.

Recordamos rápidamente, dentro de la jerarquía de objetos del navegador, cuáles son los correspondientes al objeto `form` que veremos aquí:

```
- form    <FORM> ... </FORM>
   + button     <INPUT TYPE="button">
   + checkbox   <INPUT TYPE="checkbox">
   + 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>
```

Estos serán los elementos que estudiaremos dentro de `form`.

Formularios: 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` (el array `forms`). Cada elemento de este array es un objeto de tipo `form`.

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)

`name`
: Es una cadena que contiene el nombre que se le pone al formulario en su parámetro `NAME` y vía el que podemos acceder a él desde JavaScript.

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

Con este sencillo ejemplo vemos cómo acceder a los formularios y elementos de los mismos sin necesidad de conocer sus nombres:

```
<HTML> <HEAD></HEAD>
<BODY>
<FORM ACTION="mailto:nobody@punto.es" METHOD="POST" NAME="ElPrimero">
Hola, soy el primer formulario del documento. Tengo algunos elementos: <P>

Caja 1: <INPUT TYPE=TEXT SIZE="5" MAXLENGTH="10" NAME="Linea1"><BR>
Caja 2: <INPUT TYPE=TEXT SIZE="5" MAXLENGTH="10" NAME="Linea2"><BR>
Caja 3: <INPUT TYPE=TEXT SIZE="5" MAXLENGTH="10" NAME="Linea3">
</FORM> <P>
```

```
<FORM ACTION="mailto:nobody@punto.es" METHOD="GET" NAME="ElSegundo">
Hola, soy el segundo formulario del documento. Tengo algunos elementos: <P>

<INPUT TYPE=RADIO NAME="Radio1" VALUE="a" CHECKED> Radio 1<BR>
<INPUT TYPE=RADIO NAME="Radio1" VALUE="b"> Radio 2<BR>
<INPUT TYPE=RADIO NAME="Radio1" VALUE="c"> Radio 3
</FORM> <P>

<SCRIPT LANGUAGE="JavaScript"> <!--
 var i, j;

 document.write('Tenemos ' + document.forms.length
   + ' formularios en el documento.');

 for (i = 0; i < document.forms.length; i++)
   with(document) {
     write('<P>document.forms[' + i + '].name = ' + forms[i].name + '<BR>');
     write('document.forms[' + i + '].action = ' + forms[i].action + '<BR>');
     write('document.forms[' + i + '].method = ' + forms[i].method + '<P>');
     write('Tengo ' + forms[i].elements.length + ' elementos:<P>');

     for (j = 0; j < document.forms[i].elements.length; j++)
       write('document.forms[' + i + '].elements[' + j + '].name = '
         + forms[i].elements[j].name + '<BR>');
     }

//--> </SCRIPT>

</BODY> </HTML>
```

Formularios: 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.

Propiedades de los objetos text, textarea y password

`defaultValue`
: 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`.

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

Formularios: 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.

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`.

Métodos de los objetos button

`click();`
: Realiza la acción de pulsado del botón

Formularios: 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"`.

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`.

Métodos del objeto checkbox

`click();`
: Realiza la acción de pulsado del botón

Formularios: 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.).

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 paráme tro `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`. Es más, para acceder a ellos desde JavaScript, como todos tienen el mismo `NAME`, pero distinto `VALUE`, tendremos que recorrer un array de radios en el que cada opción es el objeto radio cuyas propiedades se han comentado.

Métodos del objeto radio

`click();`
: Realiza la acción de pulsado del radio

Veamos un ejemplo que intenta aclarar el punto del array de objetos de tipo radio:

```
<HTML>
<HEAD></HEAD>
<BODY>

<FORM NAME="UnForm">
<INPUT TYPE=RADIO NAME="Grupo1" VALUE="Valor1">Radio 1.1<BR>
<INPUT TYPE=RADIO NAME="Grupo1" VALUE="Valor2">Radio 1.2<BR>
<INPUT TYPE=RADIO NAME="Grupo1" VALUE="Valor3">Radio 1.3<BR><HR>

<INPUT TYPE=RADIO NAME="Grupo2" VALUE="Valor1mas">Radio 2.1<BR>
<INPUT TYPE=RADIO NAME="Grupo2" VALUE="Valor2mas">Radio 2.2<BR>
<INPUT TYPE=RADIO NAME="Grupo2" VALUE="Valor3mas">Radio 2.3<BR>
```

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

 var i;

 with(document) {
   for (i = 0; i < UnForm.Grupo1.length; i++)
     write('<BR>Radio nº ' + i + ' del grupo 1. Valor: '
       + UnForm.Grupo1[i].value);

   for (i = 0; i < UnForm.Grupo2.length; i++)
     write('<BR>Radio nº ' + i + ' del grupo 2. Valor: '
       + UnForm.Grupo2[i].value);
   }

//-->
</SCRIPT>

</BODY>
</HTML>
```

Como estamos dentro de `document`, no es necesario que referenciemos al formulario como `document.UnForm`, basta con poner `UnForm`.

Formularios: 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.

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.

Formularios: 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.

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`.

Con esto termina la parte de los objetos del navegador. En el siguiente y último bloque finaliza la parte teórica del curso, estudiando el 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.

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.

Un evento no es más que un suceso para el que puede interesarnos definir una respuesta. Por ejemplo, ante una pulsación del ratón podemos querer avisar de algo. En este bloque vamos a ver qué posibilidades nos permite JavaScript para trabajar con eventos.

Eventos: Primeras nociones

Básicamente, la idea de cómo trabajar con eventos 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 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 siguiente tabla:

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.

Eventos: Más conceptos

En el capítulo anterior se dio una primera aproximación a los eventos en JavaScript, y se mostró cómo asociar código a eventos dentro de las directivas HTML.

Vimos que para crear un *manejador de evento* asociado a una directiva concreta, teníamos que escribir algo como esto:

```
<Etiqueta onEvento="CodigoJS;">
```

donde CodigoJS puede ser una llamada a una función nuestra, o una secuencia de sentencias.

Cuando creamos un manejador para el evento, estamos asignando al correspondiente objeto una propiedad que toma el nombre de dicho manejador, y es esta propiedad la que nos permite acceder al manejador del evento del objeto. Esto mismo es lo que nos va a permitir cambiar el código asociado al evento de un objeto. Vamos a explicar esto con un ejemplo.

Supongamos que queremos crear una página en la que, al pulsar una vez sobre un botón pase una cosa, y que cuando pulsemos otra vez (y las siguientes), pase algo completamente distinto a lo que sucedió la primera vez. Por ejemplo, tenemos un botón que al pulsarlo la primera vez nos mostraría una ventana preguntándonos el nombre, y que las siguientes veces que lo pulsáramos nos dijera "yo te conozco de algo, ¿verdad? Tú eres XXXXXX".

Hacer esto implica que de alguna manera debemos ser capaces de cambiar el código asociado al evento en el que queremos que se active la acción. ¿Cómo lo cambiamos? No debemos olvidar que el nombre de una función es una referencia a la zona de memoria en la que se encuentra, y si cambiamos al manejador del evento la referencia a una cierta función, estamos cambiando la función que se ejecutará cuando el evento suceda. Para aclarar ideas, vamos a ver un ejemplo:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<HTML>
<HEAD>
 <SCRIPT LANGUAGE="JavaScript">
 <!--
 var Nombre;

 function Pulsacion1Boton() {
   Nombre = prompt("Dime cómo te llamas", "Mi nombre");
   document.Form1.Boton1.onclick = Pulsacion2Boton;
 }

 function Pulsacion2Boton() {
   alert("Hola, \"" + Nombre + "\", creo que ya nos conocemos :-)");
 }
 //-->
 </SCRIPT>
</HEAD>
```

```
<BODY BGCOLOR=white>

<FORM NAME="Form1">
 <INPUT TYPE=BUTTON NAME="Boton1" VALUE="Púlsame" onClick="Pulsacion1Boton();">
</FORM>

</BODY>
</HTML>
```

Lo importante del ejemplo es ver que se puede asignar otra función de forma directa al manejador del evento.

Vamos a terminar este capítulo viendo nuevos eventos introducidos con la versión 1.2 de JavaScript:

Evento	Afecta a...	Sucede cuando...	Nombre del manejador
Abort	Imágenes	El usuario interrumpe la carga de una imagen	onAbort
DblClick	Enlaces, botones	El usuario hace doble click sobre el objeto	onDblClick
DragDrop	Ventanas	El usuario suelta un objeto sobre la ventana del navegador	onDragDrop
Error	Imágenes, ventanas	La carga de un documento o de una ventana resulta errónea	onError
KeyDown	Documentos, enlaces, imágenes, áreas de texto	Se produce justo en el momento en que el usuario pulsa una tecla	onKeyDown
KeyPress	Documentos, enlaces, imágenes, áreas de texto	Tras un KeyPress, mientras se mantenga pulsada la tecla	onKeyPress
KeyUp	Documentos, enlaces, imágenes, áreas de texto	Se produce justo en el momento de soltar la tecla pulsada	onKeyUp
MouseDown	Documentos, botones, enlaces	Se produce justo en el momento en que el usuario pulsa un botón del ratón	onMouseDown
MouseMove	Por defecto, ningún objeto	El usuario mueve el ratón	onMouseMove
MouseUp	Documentos, botones, enlaces	El usuario suelta el botón del ratón	onMouseUp
Move	Ventanas	El usuario (o la acción de un script) mueve una ventana	onMove
Reset	Formularios	El usuario resetea un formulario (p. ej, pulsando el botón "Reset")	onReset
Resize	Ventanas	El usuario (o la acción de un script) cambia de tamaño una ventana	onResize

Vamos a comentar algunas cuestiones sobre el evento `onError`. Este evento se da cuando hay un error de sintaxis JavaScript, NO cuando hay un error del navegador. Por ejemplo, si intentamos cargar una página que no existe, no se dará este evento.

Podremos hacer lo siguiente:

- Asignarle `null`: esto cancelará todos los diálogos de error de JavaScript
- Asignarle una función que sea su manejador. Para cancelar la ventana estándar de errores JavaScript, esta función debe devolver `true`.

Si somos nosotros quienes escribimos una función que sea el manejador del error, tenemos tres opciones para notificar los errores:

- Trazar los errores pero permitir al diálogo estándar de JavaScript que los notifique. Para ello, habrá que usar una función manejadora que devuelva `false` o que no devuelva valor alguno.
- Deshabilitar el diálogo estándar y notificar nosotros mismos los errores. Para ello, la función manejadora del evento tendrá que devolver `true`.
- Deshabilitar por completo la notificación de errores. Para ello, simplemente asignaremos `null` al manejador del evento.

En el siguiente capítulo veremos una peculiaridad del navegador NS: la captura de eventos.

Captura de eventos

Cuando se diseñó el primer modelo de eventos, se hizo de una forma relativamente sencilla: estudiar qué elementos HTML tenía sentido que reaccionaran ante ciertos eventos, y con ello surgió la primera idea vista hace dos capítulos. Sin embargo, buscando la interactividad con el usuario, se vio que este modelo de eventos era insuficiente si se quería conseguir este objetivo. Por ello, se estudió cómo ampliarlo.

En este punto, las políticas se separaron bastante: Netscape se adelantó introduciendo su objeto LAYER y la versión JavaScript 1.2. IExplorer, por contra, adoptó los criterios del W3C.

Así, si pretendemos que un objeto HTML reaccione ante un evento para el que no está pensado reaccionar en principio, en IExplorer nos basta con asociarle el manejador del evento en cuestión, mientras que con Netscape el procedimiento es más complejo. Se trata de realizar lo que se conoce como **captura de eventos**.

Como hemos comentado, por ejemplo, el objeto `window` en principio no dispone de un evento `MouseMove` o de un evento `Click`; sin embargo, sí que es cierto que sobre la ventana se mueve el ratón o se pulsa algún botón del ratón. ¿Qué haríamos si quisiéramos que, pulsáramos donde pulsáramos en la ventana, sucediera algo? Netscape nos da los siguientes métodos, aplicables a los objetos `window`, `document` y `layer`:

- `captureEvents`: captura un evento de un tipo especificado
- `releaseEvents`: ignora la captura de eventos de un tipo especificado
- `routeEvent`: encamina el evento capturado a un objeto especificado
- `handleEvent`: maneja el evento capturado (este método no pertenece a `layer`)

Los pasos a seguir para poner en marcha la captura de eventos son:

Habilitar la captura de eventos

Supongamos que queremos capturar todas las pulsaciones del ratón en la ventana. Este evento es el evento `CLICK`, así que, para habilitar la captura de todos los *click* en la ventana escribiríamos:

```
window.captureEvents(Event.CLICK);
```

Si queremos capturar más de un evento tendremos que escribirlos separados por el caracter `|` (ALT + 0124). Por ejemplo, si queremos capturar el *click* y el *mousemove*, escribiríamos:

```
window.captureEvents(Event.CLICK | Event.MOUSEMOVE);
```

El argumento que pasamos a `captureEvents` es una propiedad del objeto `Event` que indica el evento a capturar (en el siguiente capítulo estudiamos este objeto).

Definir el manejador del evento

Ahora tenemos que definir una función cuyo único parámetro va a ser una variable de tipo `Event`, en la que incluiremos el código necesario para que se realicen las acciones que queremos llevar a cabo. Seguiría un esquema como este:

```
function ManejadorEventoClick(e) {
  // Aquí va el código que quieras asociar al evento Click capturado para la ventana
```

```
   }
```

Ahora tenemos las siguientes opciones para manipular el evento, una vez definida la función:

- Devolver `true`. Si el manejador de evento fue definido para un enlace, entonces entramos al enlace y no se comprueban más manejadores de eventos (por ejemplo, de un botón o de un frame hijo).
- Devolver `false`. En el caso de un enlace, no entraremos en él. Si el evento es no cancelable, esto finaliza el manejo del evento.
- Llamar a la función `routeEvent`. En este caso, JS buscará otros manejadores para el evento. Si otro objeto intenta capturar el evento (como, por ejemplo, `document`), JS llama a su manejador. Si no hay otro objeto intentando capturarlo, JS busca el manejador para el destino original del evento. Esta función devuelve el valor devuelto por el manejador del evento. El objeto capturador puede mirar este valor de retorno y decidir cómo proceder.

  Cuando `routeEvent` llama a un manejador de evento, el manejador es activado. Si `routeEvent` llama a un manejador de evento cuya función es visualizar una nueva página, la acción se realiza sin volver al objeto capturador.

- Llamar al método `handleEvent` de un receptor de eventos. Cualquier objeto que pueda registrar manejadores de eventos es un receptor de eventos. Este método llama explícitamente al manejador de eventos del objeto receptor y se salta la jerarquía capturadora.

Por ejemplo: si queremos que todos los eventos `click` vayan al primer enlace de la página, usaríamos

```
function ManejadorClick(e) {
   window.document.links[0].handleEvents(e);
   }
```

Registrar el manejador de eventos

Finalmente, registramos la función como el manejador de evento para ese evento:

```
   window.onClick = ManejadorClick;
```

Veamos un ejemplo: Al pulsar en cualquier posición de la ventana, mostraremos un `alert` con las coordenadas de la posición.

```
function ManejarEvClick(e) {
  alert('PosX: ' + e.screenX + ' PosY: ' + e.screenY);
  }

window.captureEvents(Event.CLICK);
window.onclick = ManejarEvClick;
```

El objeto EVENT (Netscape)

Cada evento tiene asociado un objeto de tipo `event`. Este objeto nos da una serie de propiedades que nos informarán de las coordenadas del ratón en el momento de producirse el evento, el tipo de evento... Cuando sucede un evento, si existe un manejador asociado a él, el objeto `event` es enviado como argumento al manejador del evento en cuestión.

Para gracia (o desgracia) del desarrollador JavaScript, Netscape y Microsoft han seguido filosofías completamente distintas, dando lugar a dos objetos `event` más bien poco parecidos, y a dos procedimientos de captura y manejo de eventos que, para que dudarlo, son muy poco parecidos. En este capítulo veremos cuál fue la implementación del objeto `event` dada por Netscape para su navegador.

El objeto `event` contiene una serie de propiedades que describen un evento JavaScript. Este objeto es pasado como argumento a los manejadores de eventos cuando dichos eventos suceden. Además, es creado por el navegador cuando se da el evento, nunca por el programador.

Por ejemplo, en el caso de un evento `onMouseDown`, el objeto `event` contiene información tanto acerca del botón que se ha pulsado así como de qué teclas modificadoras (i.e., CTRL, ALT, SHIFT o META) tenía pulsadas el usuario en el momento en el que sucedió el evento.

En primer lugar, vamos a ver qué propiedades nos ofrece este objeto. Esto no quiere decir que todas esas propiedades estén disponibles para cualquier evento. Lo que sucede es que, dependiendo del evento que se de, tendremos disponibles unas u otras propiedades.

Propiedad	Descripción
data	Devuelve un array de cadenas que contienen las URLs de los objetos "dropped". Es pasada con el evento `DragDrop`
height	Se trata de la altura de la ventana o del frame
layerX	Este número representa el ancho del objeto cuando se trata de un evento `resize`, o la posición horizontal del cursor en pixels relativa a la capa en la que el evento sucedió
layerY	Este número representa el alto del objeto cuando se trata de un evento `resize`, o la posición vertical del cursor en pixels relativa a la capa en la que el evento sucedió
modifiers	Cadena especificando las teclas modificadoras asociadas a un evento de teclado o de ratón. Los valores posibles son: `ALT_MASK`, `CONTROL_MASK`, `SHIFT_MASK` y `META_MASK`
pageX	Representa la posición horizontal del cursor en píxels, relativa a la página
pageY	Representa la posición vertical del cursor en píxels, relativa a la página
screenX	Representa la posición horizontal del cursor en píxels, relativa a la pantalla
screenY	Representa la posición vertical del cursor en píxels, relativa a la pantalla
target	Cadena que representa al objeto para el cual el evento fue enviado originalmente. Esta propiedad la encontraremos en cualquiera de los eventos
type	Cadena representando el tipo de evento
which	Si se trata de un evento de ratón, entonces tendremos los siguientes valores: 1 -> Se ha pulsado el botón izquierdo, 2 -> Se ha pulsado el botón central, 3 -> Se ha pulsado el botón derecho Si se trata de un evento de teclado, contiene el código ASCII de la tecla pulsada
width	Representa el ancho de la ventana o de un frame
x	Igual que layerX
y	Igual que layerY

Vamos a ver un pequeño ejemplo en el que asociamos un manejador de evento al objeto document, para el evento onMouseDown, de manera que cuando pulsemos con el ratón sobre cualquier parte del documento, se nos muestre alguna información de los campos del objeto event que se creará para esta situación:

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

function DimeAlgo(e) {
  alert('Se ha detectado un evento para el objeto document\n'
    + 'La naturaleza de este evento es: ' + e.type + '\n'
    + 'La posición x del cursor era: ' + e.layerX + '\n'
    + 'La posición y del cursor era: ' + e.layerY + '\n');
    + 'Y el valor de modifiers: ' + e.modifiers);

  return true;
  }

document.onmousedown = DimeAlgo;

//-->
</SCRIPT>
```

Vista la tabla y el ejemplo, vamos a ver ahora a qué propiedades de event podemos acceder según los distintos eventos.

onAbort, onBlur, onChange
: Tendremos acceso a type, target

onClick, onDblClick, onMouseDown, onMouseUp
: Tendremos acceso a type, target, which y modifiers. Además, cuando se pulse un enlace tendremos acceso a layerX, layerY, pageX, pageY, screenX, screenY.

  En which tendremos uno de los siguientes valores: 1, para el botón izquierdo del ratón y 3, para el botón derecho del ratón. En modifiers tendremos la lista de teclas modificadoras que estaban pulsadas cuando sucedió el evento.

  Si el manejador devuelve false, la acción por defecto del objeto es cancelada como sigue:

  - Los botones no tienen acción por defecto: no sucede nada
  - Radio buttons y checkboxes: no se marcan
  - Botón SUBMIT: no se envía el form
  - Botón RESET: no se borra el form

onDragDrop
: Tendremos acceso a type, target, data, modifiers, screenX y screenY. En data tendremos un array de cadenas con las URLs de los objetos "dropped"

onError, onFocus
: Tendremos acceso a type, target.

`onKeyDown`, `onKeyPress`, `onKeyUp`
Tendremos acceso a `type`, `target`, `layerX`, `layerY`, `pageX`, `pageY`, `screenX`, `screenY`, `which`, `modifiers`

En `which` tendremos el código ASCII de la tecla pulsada. Para obtener la tecla en sí, usaremos el método `String.fromCharCode`. En `modifiers` tendremos una lista con las teclas modificadoras pulsadas.

`onLoad`
Tendremos acceso a `type`, `target` `width`, `height` (de la ventana si es un evento sobre una ventana y no sobre un layer).

`onMouseMove`
Tendremos acceso a `type`, `target`, `layerX`, `layerY`, `pageX`, `pageY`, `screenX`, `screenY`

`onMouseOut` (el ratón abandona una área o el área de un link)
Tendremos acceso a `type`, `target`, `layerX`, `layerY`, `pageX`, `pageY`, `screenX`, `screenY`

`onMove` (sucede un evento de movimiento: el usuario o un script mueve una ventana o frame)
Tendremos acceso a `type`, `target`, `screenX`, `screenY`. Estas dos últimas nos dan la posición superior izquierda de la ventana.

`onReset`
Tendremos acceso a `type`, `target`

`onResize`
Tendremos acceso a `type`, `target`, `width`, `height`. Estas dos últimas nos dan el ancho y alto de la ventana o el frame.

`onSelect` (seleccionar texto de un `text` o un `textarea`)
Tendremos acceso a type, target

`onSubmit`
Tendremos acceso a `type`, `target`

Nota: Para enviar un form por `mailto:` o por `news:` se requiere el privilegio `UniversalSendMail`. Ver la Client-Side JavaScript Guide (podeis encontrar el enlace en la sección de enlaces) para más detalles.

`onUnload`
Tendremos acceso a `type`, `target`

Ejemplos específicos para Netscape sobre captura de eventos

Interceptamos un error por llamar a una función que no existe:

```
<HTML> <HEAD>
 <SCRIPT LANGUAGE="JavaScript"> <!--
 function ManejarError(e) {
   alert('Este evento iba dirigido a ' + e.target + ', pero\n'
     + 'lo he cazado yo. Disculpen las molestias');
   return true;
   }
 onerror = ManejarError;
 //--> </SCRIPT>
 </HEAD>
```

```
<BODY>
<SCRIPT LANGUAGE="JavaScript">
<!--

 NoExisto();

//-->
</SCRIPT>

</BODY>
</HTML>
```

No permitimos pulsar algunas teclas en un formulario:

```
<HTML> <HEAD>
 <SCRIPT LANGUAGE="JavaScript"> <!--
 function NoQuieroAB(e) {
   var Tecla = String.fromCharCode(e.which);
   if(
         Tecla.toUpperCase() == 'A'
      || Tecla.toUpperCase() == 'B'
     )
     return false;
   else return true;
   }
 //--> </SCRIPT>
 </HEAD>
 <BODY>
 <FORM NAME="UnForm">
  Escribe algo: <INPUT TYPE=TEXT SIZE=20 MAXLENGTH=25 NAME="LineaTexto">
 </FORM>
 <SCRIPT LANGUAGE="JavaScript"> <!--
   document.UnForm.LineaTexto.onkeypress = NoQuieroAB;
 //--> </SCRIPT>

 </BODY>
 </HTML>
```

El objeto EVENT (IExplorer)

Finalizamos el estudio de los eventos viendo el objeto event que nos ofrece el navegador Internet Explorer a partir de la versión 4. Aunque la forma de llevarlo a cabo sea distinta, la idea que hay por debajo es la misma que había para Netscape: representa el estado de un evento, es decir, qué tipo de circunstancias se dieron cuando el evento sucedió, como pulsación de alguna tecla, el ratón en alguna posición, etc.

El objeto event está disponible únicamente mientras se da un evento. Esto quiere decir que podremos usarlo dentro de un manejador de evento, pero no en cualquier otra parte del código. A pesar de que todas las propiedades de event están disponibles, hay algunas propiedades que sólo tienen sentido cuando se da un determinado evento y no otro.

Al contrario que sucede con Netscape, con Explorer no es necesario realizar captura de eventos alguna, ni hay que pasar a las funciones manejadoras el parámetro del evento. En lugar de ello, directamente tratamos con event.Propiedad en la función manejadora.

Pasamos a ver las propiedades más destacadas que nos ofrece este objeto. Para más información con el resto de propiedades, podeis consultar la referencia MSDN cuyo enlace aparece en la sección de enlaces.

Propiedad	Descripción breve
altKey	Nos dice el estado de la tecla ALT
altLeft	Nos dice el estado de la tecla ALT izquierda
button	Nos dice qué botón ha sido pulsado
clientX	Nos dice la coordenada x del ratón relativa al área cliente de la ventana, excluyendo barras de scroll y otros elementos decorativos
clientY	Idem para la coordenada y del ratón
ctrlKey	Nos dice el estado de la tecla CTRL
ctrlLeft	Nos dice el estado de la tecla LEFT izquierda
keyCode	Nos dice el código Unicode asociado a la tecla que causó el evento
offsetX	Nos dice la coordenada x del ratón relativa al objeto que lanzó el evento
offsetY	Idem para la coordenada y del ratón
repeat	Nos dice si el evento onKeyDown está siendo repetido
returnValue	Podremos asignar un valor de retorno para el evento asignándolo a esta propiedad
screenX	Nos dice la coordenada x del ratón relativa a la pantalla del usuario
screenY	Idem para la coordenada y del ratón
shiftKey	Nos dice el estado de la tecla SHIFT
shiftLeft	Nos dice el estado de la tecla SHIFT izquierda
toElement	Nos da una referencia del objeto hacia el que el usuario está moviendo el ratón
type	Nos da el nombre del evento
x	Nos da la coordenada x del ratón (en pixels) relativa al elemento padre
y	Idem con la coordenada y del ratón

Veremos algún ejemplo para Explorer en el apartado de ejercicios resueltos.

Ejercicios propuestos

En este apartado se recoge una variada colección de ejercicios propuestos para resolver. Completándolos se conseguirá un mayor dominio del lenguaje. Si teneis alguna duda o problema al resolverlos, no dudeis en escribirme.

Los ejercicios están agrupados en los siguientes bloques:

No enviaré las soluciones por e-mail: espero que los resolvais vosotros ;-)

Ejercicios: Sintaxis

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 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.
3. Escribid una función que calcule y muestre la suma de los 1000 primeros números naturales.
4. 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.
5. 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.
6. Usando la función `prompt`, solicitar un número, y mostrar todos los números primos menores o iguales que el que entre el usuario.

Ejercicios: Objetos

1. 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.
2. 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. Modificar 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.

Ejercicios: Objetos del lenguaje

1. Diseñar una función que sume/reste matrices y muestre el resultado de alguna de estas operaciones por pantalla.
2. Crear una función separadora de cadenas, de acuerdo a la siguiente especificación:

   Será llamada con tres parámetros: la cadena, el carácter separador y la posición. La cadena es la cadena que queremos tratar. El carácter separador nos dice cuál va a ser el carácter que busquemos para separar la cadena en trocos. La posición es el número de trozo que resulta de esa separación.

   Por ejemplo, tenemos la cadena "Hola a todos" y queremos que el separador sea el espacio. Troceando la cadena por este separador, obtendríamos tres subcadenas: "Hola", "a", "todos". Ahora, con la posición decimos cuál de estos trozos queremos. Si la posición es 1, queremos el trozo "Hola", si es 2, "a", y si es 3, "todos". Cualquier otra posición devolvería como subcadena la vacía. Si la cadena fuera "Hola a ", y separando por espacios queremos el tercer trozo, el resultado sería la cadena vacía.

   No vale usar la función `split`: el ejercicio consiste en descubrir cómo codificarla.

3. Crear un formulario en el que se introduzcan datos de fecha de nacimiento y, a partir del día actual, se nos informe de cuántos años tenemos, cuántos días llevamos vivos, minutos, segundos, etc.
4. Crear un reloj con la hora actual que actualice la hora mostrada cada cinco segundos.
5. Encontré en la página http://www.nyx.net/~gthompso/ un interesante trozo de código, obra de Geoffrey A Swift. El ejercicio consiste en averiguar, sin ejecutarlo, qué hace este código:

```
a=new Array();a[0]='a=new Array();'; a[1]='['; a[2]=']';
a[3]='\''; a[4]='\\'; a[5]='='; a[6]='a'; a[7]=';'; a[8]='';
a[9]='for(i=0;i<10;i++)document.writeln((i==0?a[0]:a[8])+a[6]+a[1]+i+a[2]+a[5]+a[3]+(
(i==3||i==4)?a[4]:a[8])+a[i]+a[3]+a[7]+(i==9?a[9]:a[8]))';
for(i=0;i<10;i++)document.writeln((i==0?a[0]:a[8])+a[6]+a[1
]+i+a[2]+a[5]+a[3]+((i==3||i==4)?a[4]:a[8])+a[i]+
a[3]+a[7]+(i==9?a[9]:a[8]))
```

Ejercicios: Objetos del navegador

Vamos a aprovechar que llegados a este punto ya se ha visto el objeto `form`, para proponer una colección de ejercicios que pueden ser muy básicos, pero que incluyen el uso de formularios para resolverlos.

1. Dados dos valores introducidos por el usuario en un formulario, mostrar la suma, resta, producto y división de ambos.
2. Extender el ejercicio y realizar una calculadora.
3. Seguimos con calculadoras, pero esta más concreta: se trata de elegir una figura geométrica y, en función de la figura, coger los parámetros necesarios para calcular su área y longitud, mostrando el resultado en un `text`.
4. Crear un formulario que de a elegir todos los parámetros de apertura de una ventana desde JavaScript, junto con un botón "Crear Script" que al pulsarlo introduzca dentro de un `textarea` el código necesario para crear el script.
5. Usando frames, crear un proyector. La idea será dividir la pantalla en tres zonas. Tendremos un frame izquierdo donde se especificará la dimensión de las imágenes a mostrar así como el número de fotogramas de la película, dando valores por defecto. Tendremos un frame donde se mostrarán las imágenes (una por una), y tendremos un último frame en la parte de abajo con controles de marcha y paro de la película, avance de fotogramas de uno en uno, pausa y vuelta al principio. Las imágenes deben precargarse.

Ejercicios: Eventos

1. Ampliar la validación de un formulario para que en los campos que deban aceptar números positivos, no admitan otro carácter. Esto quiere decir que cuando introduzcan un carácter, nosotros debemos ver cuál es el carácter introducido, para comprobar si es o no numérico y rechazarlo en ese caso.
2. Completar el ejemplo resuelto número 10 ("Arrastrar y soltar") para que también funcione con Netscape.
3. Completar el ejemplo resuelto número 11 ("Menús desplegables") para que también funcione con Netscape 6.

Referencia abreviada: Sintaxis y funciones

Sintaxis (I): Inclusión de código y generalidades

Usando <SCRIPT>

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

Como respuesta a eventos

```
<DIRECTIVA nombreEvento="Código_JavaScript">
```

Generalidades

- Es "Case Sensitive", es decir, distingue mayúsculas de minúsculas.
- Comentarios: `/* ... */` para encerrar un bloque y `//` para comentarios de una línea.
- Cada sentencia ha de terminar en `;`
- Encerrando código entre llaves `{ ... }` lo agrupamos. Se verá su sentido cuando tratemos las estructuras de control.

Sintaxis (I): Variables y constantes

Variables

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

Ámbito 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>`

Constantes

Las constantes no existen como tales en JavaScript, salvando el caso del objeto `Math`.

Sintaxis (I): Tipos de datos

Enteros
Flotantes
Booleanos
Nulos
Indefinidos
Cadenas
Objetos
Funciones

En las cadenas, podemos usar los caracteres de escape que tenemos en C para representar los saltos de línea, tabulaciones, etc...

```
\b   Espacio hacia atrás
\f   Alimentación de línea
\n   Nueva línea
\r   Retorno de carro
\t   Tabulación
\\   Barra invertida: \
\'   Comilla simple: '
\"   Comilla doble: "
```

Operadores

El operador más básico es el operador unario de asignación, =. Cuando estamos utilizando cadenas, el operador + tiene el significado **"concatenación"**.

Operadores: Aritméticos

```
+     Suma
-     Resta
*     Producto
/     Cociente
%     Módulo
```

Abreviaturas:

```
+=     b += 3  equivale a   b = b + 3
-=     b -= 3  equivale a   b = b - 3
*=     b *= 3  equivale a   b = b * 3
/=     b /= 3  equivale a   b = b / 3
%=     b %= 3  equivale a   b = b % 3
```

Incrementos y decrementos: ++ y --.

Por último, dentro de los operadores aritméticos tenemos el operador unario -, que aplicado delante de una variable, le cambia el signo.

Operadores: Comparación

```
==        igual
!=        distinto
>         mayor que
<         menor que
>=        mayor o igual que
<=        menor o igual que
```

A partir de JavaScript 1.3 esta lista se amplia con dos operadores más:

```
===        estrictamente igual
!==        estrictamente distinto
```

Operadores: Lógicos

```
 &&     AND ('y' lógica)
 ||     OR ('o' lógica)
 !      NOT ('no' lógica)
```

Operadores: A nivel de bit

```
&        AND bit a bit
|        OR  bit a bit
^        XOR bit a bit
~        NOT bit a bit
>>       rotación a derecha
<<       rotación a izquierda
```

Operadores: Resto

```
 new, delete, void, typeof
```

Sintaxis (II): Condicionales

El condicional `if`

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

También tenemos el condicional ternario:

```
[ Expresión ] ? [ Sentencia1 ] : [ Sentencia2 ] ;
```

El condicional switch

```
switch(condicion) {
  case caso1 :
    sentencias para caso1;
    break;
       ......
  case casoN :
    sentencias para casoN;
    break;
  default :
    sentencias por defecto;
    break;
  }
```

Sintaxis (II): Bucles

El bucle for

```
for([Inicialización]; [Condición]; [Expresión de actualización])
  {
    Instrucciones a repetir
  }
```

El bucle while

```
while(Condición) {
 Instrucciones a repetir
 }
```

El bucle do ... while

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

El bucle for ... in

```
for(var 'Propiedad' in 'Objeto') {
  Instrucciones a repetir
  }
```

La construcción with

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

Sintaxis (II): Funciones

```
function NombreFuncion(arg1, ..., argN)  {
  Código de la función


  return Valor;
  }
```

Las funciones en JavaScript tienen una propiedad particular, y es que no tienen un número fijo de argumentos. Los argumentos pueden ser accedidos bien por su nombre, bien por un vector de argumentos que tiene asociada la función (que será NombreFuncion.arguments), y podemos saber cuántos argumentos se han entrado viendo el valor de NombreFuncion.arguments.length

Funciones del lenguaje

```
parseInt(cadena,base);
parseFloat(cadena);
escape(cadena);
unescape(códigos);
isNaN(valor);
eval(expresión);
```

Objetos: Cómo son en JavaScript

Crear nuestros propios moldes para objetos:

```
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;
  }
```

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 };
```

Referencia abreviada: Objetos del lenguaje

Objetos del lenguaje: String

Propiedades del objeto String

```
length
prototype
```

Métodos del objeto String

```
anchor(nombre)
big()
blink()
charAt(indice)
fixed()
fontcolor(color)
fontsize(tamaño)
indexOf(cadena_buscada, indice)
italics()
lastIndexOf(cadena_buscada, indice)
link(URL)
small()
split(separador)
strike()
sub()
substring(primer_Indice, segundo_Indice)
sup()
toLowerCase()
toUpperCase()
```

Objetos del lenguaje: Array

Para poder usar un objeto array, tendremos que crearlo con su constructor, por ejemplo:

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

Propiedades del objeto Array

```
length
prototype
```

Métodos del objeto Array

```
join(separador)
reverse()
sort()
```

Objetos del lenguaje: Math

Propiedades del objeto Math

```
E
LN2
LN10
LOG2E
LOG10E
PI
SQRT1_2
SQRT2
```

Métodos del objeto Math

```
abs(numero)
acos(numero)
asin(numero)
atan(numero)
atan2(x,y)
ceil(numero)
cos(numero)
exp(numero)
floor(numero)
log(numero)
max(x,y)
min(x,y)
pow(base,exp)
random()
round(numero)
sin(numero)
sqrt(numero)
tan(numero)
```

Objetos del lenguaje: Date

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.

Métodos del objeto Date

```
getDate();
getDay();
getHours();
getMinutes();
getMonth();
getSeconds();
getTime();
getYear();
setDate(día_mes);
setDay(día_semana);
setHours(horas);
setMinutes(minutos);
setMonth(mes);
setSeconds(segundos);
setTime(milisegundos);
setYear(año);
toGMTString();
```

Objetos del lenguaje: Number

Propiedades, sólo accesibles desde `Number`:

```
MAX_VALUE
MIN_VALUE
NaN
NEGATIVE_INFINITY
POSITIVE_INFINITY
```

Objetos del lenguaje: Boolean

Este objeto nos permite crear booleanos, tomando los valores `'true'` o `'false'`. Podemos crear objetos de este tipo mediante su constructor.

Objetos del lenguaje: Function

Nos proporciona la propiedad `'arguments'`, 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.

Referencia abreviada: Objetos del navegador

Los objetos del navegador: Jerarquía

```
* 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
```

Objetos del navegador: window

Propiedades del objeto window

```
closed
defaultStatus
frames
history
length
location
name
opener
parent
self
status
top
window
```

Métodos del objeto window

```
alert(mensaje)
blur()
clearInterval(id)
clearTimeout(nombre)
close()
confirm(mensaje)
focus()
moveBy(x, y)
moveTo(x, y)
```

```
open(URL, nombre, características)
```

- 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
- outerWidth = px
- outerHeight = px
- left = px
- top = px

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

```
prompt(mensaje,respuesta_por_defecto)
scroll(x, y)
scrollBy(x, y)
scrollTo(x, y)
setInterval(expresion, tiempo)
setTimeout(expresion, tiempo)
```

Objetos del navegador: frame

Propiedades del objeto frame

```
frames
parent
self
top
window
```

Métodos del objeto frame

```
alert(mensaje)
blur()
clearInterval(id)
clearTimeout(nombre)
close()
confirm(mensaje)
focus()
moveBy(x, y)
moveTo(x, y)
open(URL, nombre, características)
```

Características:

```
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)
```

Objetos del navegador: location

Recordemos que la sintaxis de una URL era:

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

Propiedades del objeto location

```
hash
host
hostname
href
pathname
port
protocol
search
```

Métodos del objeto location

```
reload()
replace(cadenaURL)
```

Objetos del navegador: history

Propiedades del objeto history

```
current
next
length
previous
```

Métodos del objeto history

```
back()
forward()
go(posición)
```

Objetos del navegador: navigator

Propiedades del objeto navigator

```
appCodeName
appName
appVersion
language (NS), systemLanguage (IE)
mimeTypes
platform
plugins
userAgent
```

Métodos del objeto navigator

```
javaEnabled()
```

Objetos del navegador: document

Propiedades del objeto document

```
alinkColor
anchors
applets
bgColor
cookie
domain
embeds
fgColor
forms
images
lastModified
linkColor
links
location
referrer
title
vlinkColor
```

Métodos del objeto document

```
clear();
close();
open(mime, "replace");
write();
writeln();
```

Objetos del navegador: link

Propiedades del objeto link

```
target
hash
host
hostname
href
pathname
port
protocol
search
```

Objetos del navegador: anchor

Propiedades del objeto anchor

```
href
target
```

Objetos del navegador: image

Propiedades del objeto image

```
border
complete
height
hspace
lowsrc
name
prototype
src
vspace
width
```

Los objetos del navegador: Formularios

Recordamos rápidamente, dentro de la jerarquía de objetos del navegador, cuáles son los correspondientes al objeto `form` que veremos aquí:

```
- form    <FORM> ... </FORM>
   + button     <INPUT TYPE="button">
   + checkbox   <INPUT TYPE="checkbox">
   + 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>
```

Formularios: objeto form

Propiedades del objeto form

```
action
elements
encoding
method
name
```

Métodos del objeto form

```
reset();
submit();
```

Formularios: objetos text, textarea y password

Propiedades de los objetos text, textarea y password

```
defaultValue
name
value
```

Métodos de los objetos text, textarea y password

```
blur();
focus();
select();
```

Formularios: objetos button

Propiedades de los objetos button

```
name
value
```

Métodos de los objetos button

```
click();
```

Formularios: objeto checkbox

Propiedades del objeto checkbox

```
checked
defaultChecked
name
value
```

Métodos del objeto checkbox

```
click();
```

Formularios: objeto radio

Propiedades del objeto radio

```
checked
defaultChecked
length
name
value
```

Hay que recordar que para agrupar elementos de tipo radio, todos ellos deben tener el mismo valor en su parámetro NAME. Es más, para acceder a ellos desde JavaScript, como todos tienen el mismo NAME, pero distinto VALUE, tendremos que recorrer un array de radios en el que cada opción es el objeto radio cuyas propiedades se han comentado.

Métodos del objeto radio

```
click();
```

Formularios: objeto select

Propiedades del objeto select

`length`
`name`
`options`
Array que contiene cada una de las opciones de la lista:

- `defaultSelected`
- `index`
- `length`
- `options`
- `selected`
- `text`
- `value`

`selectedIndex`

Formularios: objeto hidden

Propiedades del objeto hidden

`name`
`value`

Referencia abreviada: Eventos

Eventos: Primeras nociones

Eventos básicos:

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>

Eventos: Más conceptos

Nuevos eventos introducidos con la versión 1.2 de JavaScript:

Evento	Afecta a...	Sucede cuando...	Nombre del manejador
Abort	Imágenes	El usuario interrumpe la carga de una imagen	onAbort
DblClick	Enlaces, botones	El usuario hace doble click sobre el objeto	onDblClick
DragDrop	Ventanas	El usuario suelta un objeto sobre la ventana del navegador	onDragDrop
Error	Imágenes, ventanas	La carga de un documento o de una ventana resulta errónea	onError
KeyDown	Documentos, enlaces, imágenes, áreas de texto	Se produce justo en el momento en que el usuario pulsa una tecla	onKeyDown
KeyPress	Documentos, enlaces, imágenes, áreas de texto	Tras un KeyPress, mientras se mantenga pulsada la tecla	onKeyPress
KeyUp	Documentos, enlaces, imágenes, áreas de texto	Se produce justo en el momento de soltar la tecla pulsada	onKeyUp
MouseDown	Documentos, botones, enlaces	Se produce justo en el momento en que el usuario pulsa un botón del ratón	onMouseDown
MouseMove	Por defecto, ningún objeto	El usuario mueve el ratón	onMouseMove
MouseUp	Documentos, botones, enlaces	El usuario suelta el botón del ratón	onMouseUp
Move	Ventanas	El usuario (o la acción de un script) mueve una ventana	onMove
Reset	Formularios	El usuario resetea un formulario (p. ej, pulsando el botón "Reset")	onReset
Resize	Ventanas	El usuario (o la acción de un script) cambia de tamaño una ventana	onResize

Captura de eventos

Netscape nos da los siguientes métodos, aplicables a los objetos `window`, `document` y `layer`:

- `captureEvents`: captura un evento de un tipo especificado
- `releaseEvents`: ignora la captura de eventos de un tipo especificado
- `routeEvent`: encamina el evento capturado a un objeto especificado
- `handleEvent`: maneja el evento capturado (este método no pertenece a `layer`)

Pasos a seguir para poner en marcha la captura de eventos son:

Habilitar la captura de eventos

Definir el manejador del evento

Registrar el manejador de eventos

El objeto EVENT (Netscape)

El objeto event contiene una serie de propiedades que describen un evento JavaScript. Este objeto es pasado como argumento a los manejadores de eventos cuando dichos eventos suceden. Además, es creado por el navegador cuando se da el evento, nunca por el programador.

Vamos a ver qué propiedades nos ofrece este objeto.

Propiedad	Descripción
data	Devuelve un array de cadenas que contienen las URLs de los objetos "dropped". Es pasada con el evento DragDrop
height	Se trata de la altura de la ventana o del frame
layerX	Este número representa el ancho del objeto cuando se trata de un evento resize, o la posición horizontal del cursor en pixels relativa a la capa en la que el evento sucedió
layerY	Este número representa el alto del objeto cuando se trata de un evento resize, o la posición vertical del cursor en pixels relativa a la capa en la que el evento sucedió
modifiers	Cadena especificando las teclas modificadoras asociadas a un evento de teclado o de ratón. Los valores posibles son: ALT_MASK, CONTROL_MASK, SHIFT_MASK y META_MASK
pageX	Representa la posición horizontal del cursor en píxels, relativa a la página
pageY	Representa la posición vertical del cursor en píxels, relativa a la página
screenX	Representa la posición horizontal del cursor en píxels, relativa a la pantalla
screenY	Representa la posición vertical del cursor en píxels, relativa a la pantalla
target	Cadena que representa al objeto para el cual el evento fue enviado originalmente. Esta propiedad la encontraremos en cualquiera de los eventos
type	Cadena representando el tipo de evento
which	Si se trata de un evento de ratón, entonces tendremos los siguientes valores: 1 -> Se ha pulsado el botón izquierdo, 2 -> Se ha pulsado el botón central, 3 -> Se ha pulsado el botón derecho Si se trata de un evento de teclado, contiene el código ASCII de la tecla pulsada
width	Representa el ancho de la ventana o de un frame
x	Igual que layerX
y	Igual que layerY

Veamos a qué propiedades de event podemos acceder según los distintos eventos.

onAbort, onBlur, onChange
: Tendremos acceso a type, target

onClick, onDblClick, onMouseDown, onMouseUp
: Tendremos acceso a type, target, which y modifiers. Además, cuando se pulse un enlace tendremos acceso a layerX, layerY, pageX, pageY, screenX, screenY.

En which tendremos uno de los siguientes valores: 1, para el botón izquierdo del ratón y 3, para el botón derecho del ratón. En modifiers tendremos la lista de teclas modificadoras que estaban pulsadas cuando sucedió el evento.

Si el manejador devuelve `false`, la acción por defecto del objeto es cancelada como sigue:

- Los botones no tienen acción por defecto: no sucede nada
- Radio buttons y checkboxes: no se marcan
- Botón SUBMIT: no se envía el form
- Botón RESET: no se borra el form

`onDragDrop`
: Tendremos acceso a `type`, `target`, `data`, `modifiers`, `screenX` y `screenY`. En `data` tendremos un array de cadenas con las URLs de los objetos "dropped"

`onError`, `onFocus`
: Tendremos acceso a `type`, `target`.

`onKeyDown`, `onKeyPress`, `onKeyUp`
: Tendremos acceso a `type`, `target`, `layerX`, `layerY`, `pageX`, `pageY`, `screenX`, `screenY`, `which`, `modifiers`

En `which` tendremos el código ASCII de la tecla pulsada. Para obtener la tecla en sí, usaremos el método `String.fromCharCode`. En `modifiers` tendremos una lista con las teclas modificadoras pulsadas.

`onLoad`
: Tendremos acceso a `type`, `target` `width`, `height` (de la ventana si es un evento sobre una ventana y no sobre un layer).

`onMouseMove`
: Tendremos acceso a `type`, `target`, `layerX`, `layerY`, `pageX`, `pageY`, `screenX`, `screenY`

`onMouseOut` (el ratón abandona una área o el área de un link)
: Tendremos acceso a `type`, `target`, `layerX`, `layerY`, `pageX`, `pageY`, `screenX`, `screenY`

`onMove` (sucede un evento de movimiento: el usuario o un script mueve una ventana o frame)
: Tendremos acceso a `type`, `target`, `screenX`, `screenY`. Estas dos últimas nos dan la posición superior izquierda de la ventana.

`onReset`
: Tendremos acceso a `type`, `target`

`onResize`
: Tendremos acceso a `type`, `target`, `width`, `height`. Estas dos últimas nos dan el ancho y alto de la ventana o el frame.

`onSelect` (seleccionar texto de un `text` o un `textarea`)
: Tendremos acceso a type, target

`onSubmit`
: Tendremos acceso a `type`, `target`

Nota: Para enviar un form por `mailto:` o por `news:` se requiere el privilegio `UniversalSendMail`. Ver la Client-Side JavaScript Guide (podeis encontrar el enlace en la sección de enlaces) para más detalles.

`onUnload`
: Tendremos acceso a `type`, `target`

El objeto EVENT (IExplorer)

Propiedades más destacadas que nos ofrece este objeto:

Propiedad	Descripción breve
altKey	Nos dice el estado de la tecla ALT
altLeft	Nos dice el estado de la tecla ALT izquierda
button	Nos dice qué botón ha sido pulsado
clientX	Nos dice la coordenada x del ratón relativa al área cliente de la ventana, excluyendo barras de scroll y otros elementos decorativos
clientY	Idem para la coordenada y del ratón
ctrlKey	Nos dice el estado de la tecla CTRL
ctrlLeft	Nos dice el estado de la tecla LEFT izquierda
keyCode	Nos dice el código Unicode asociado a la tecla que causó el evento
offsetX	Nos dice la coordenada x del ratón relativa al objeto que lanzó el evento
offsetY	Idem para la coordenada y del ratón
repeat	Nos dice si el evento onKeyDown está siendo repetido
returnValue	Podremos asignar un valor de retorno para el evento asignándolo a esta propiedad
screenX	Nos dice la coordenada x del ratón relativa a la pantalla del usuario
screenY	Idem para la coordenada y del ratón
shiftKey	Nos dice el estado de la tecla SHIFT
shiftLeft	Nos dice el estado de la tecla SHIFT izquierda
toElement	Nos da una referencia del objeto hacia el que el usuario está moviendo el ratón
type	Nos da el nombre del evento
x	Nos da la coordenada x del ratón (en pixels) relativa al elemento padre
y	Idem con la coordenada y del ratón

Referencia abreviada: Colores

Esta referencia no tiene nada que ver con las anteriores. No es más que una tabla con algunas constantes que definen colores y, al lado, el color al que se hacen referencia. Es útil para poder elegir rápidamente algún color de los que está declarado en esas constantes sin necesidad de obtener el código RGB hexadecimal que define al color.

Constante	Color
aqua	
black	
blue	
fuchsia	
gray	
green	
lime	
maroon	
navy	
olive	
purple	
red	
silver	
teal	
white	
yellow	

Ejemplos resueltos

En este apartado vamos a desarrollar algunos ejemplos en JavaScript sencillos pero vistosos o útiles. En la medida de lo posible se intentará no recurrir a las nuevas adquisiciones que introdujo el HTML dinámico, aunque si aparece alguna de ellas, se nombrará para que pueda consultarse sobre ella en los múltiples documentos existentes en la red, comenzando por las referencias técnicas.

No es porque no quiera poner ejemplos tan vistosos como los que permite el HTML dinámico, es porque el curso es de JavaScript tal y como fue concebido en sus inicios. Ya el último capítulo ha tenido una bifurcación separando contenido por navegadores (el objeto `event`), y no es esta la idea cuando se trata de llevar a cabo un estándar y un conocimiento que valga para todos los navegadores que sepan interpretar JavaScript.

Si escribo demasiados ejemplos con HTML dinámico, se corre el riesgo de que quienes leáis esto os limitéis a copiar código en lugar de entender conceptos y aprender formas de hacer tareas (porque se aprende estudiando, que no copiando, el código de otros). De todas formas, incluyo algún ejemplo para que veáis que esta nueva técnica de verdad permite hacer páginas interactivas, más cómodas y entretenidas de navegar, y con facilidades extra para todos vuestros visitantes.

Y me dejo de rollos, que seguro que ya no estaréis leyendo esta línea y os habréis ido directos a ver los ejemplos ;-)

Ejemplo 1: Efecto rollover

En qué consiste

Se denomina *roll-over* al efecto de hacer que una imagen cambie por otra cuando pasa sobre ella el ratón.

La idea para conseguir este efecto sería cambiar el atributo src de la imagen por el de la nueva cuando pasa el ratón sobre ella, y restaurarla cuando el ratón deja de pasar sobre ella. Sin embargo, existe un problema: la directiva <IMG> no admite los eventos onMouseOver y onMouseOut en versiones antiguas de los navegadores, así que no será tan sencillo como <IMG onMouseOver = "CambiaImagen();return true;">. Es más, Explorer 3 ni siquiera contempla el objeto Image.

Llevándolo a cabo

Lo que se suele hacer es rodear a la imagen por medio de una directiva <A HREF>, y darle a la propia imagen el parámetro BORDER=0, para que no se rodee del borde azul del enlace. Además, para no molestar al visitante cargando la imagen por la que cambiará la existente cuando pase el ratón sobre ella, se deben precargar ambas, teniéndolas así disponibles para que este efecto sea fluido.

Precargar las imágenes es sencillo. Para precargar la imagen que se mostrará al pasar el ratón sobre la primera, debemos crear un nuevo objeto Image y darle como valor al atributo src el URL de la imagen que aparecerá. Esto podemos hacerlo en una función que usemos en la etiqueta BODY, así se cargarán las imágenes en el momento en que se carga la página. Por ejemplo, si a nuestra función la llamamos Precargar_Imagenes();, debemos poner <BODY onLoad = "Precargar_Imagenes();">. A continuación, escribimos la directiva <A HREF> y dentro la directiva <IMG>. Esta directiva tendrá los siguientes eventos: onMouseOver = "Rollover_Simple(true);" onMouseOut = "Rollover_Simple(false);", es decir:

```
<A HREF="pagina.html" onMouseOver="Rollover_Simple(true);"
 onMouseOut="Rollover_Simple(false);"><IMG SRC="IMAGEN_0.GIF"
 ID="IMAGEN" BORDER=0></A>
```

Falta ver cómo son las funciones de precargar y la que lleva a cabo el efecto propiamente dicho. Como vereis, son muy simples:

```
   function Rollover_Simple(estado)  {
     if(!estado)          document.IMAGEN.src = "IMAGEN_0.GIF";
     else         document.IMAGEN.src = "IMAGEN_1.GIF";
     }


   function Precargar_Imagenes()  {
     GRAFICOS_IMAGEN = new Image();
     GRAFICOS_IMAGEN.src = "IMAGEN_1.GIF";
     }
```

Hemos dado a las imágenes los nombres IMAGEN_0.GIF e IMAGEN_1.GIF. Estos nombres no tienen por qué ser así obviamente. Lo que debeis notar es que comparte el ID. Esto es lógico, dado que se refiere a la misma imagen, a la que posteriormente se le cambia una propiedad.

La función Precargar_Imagen() crea el objeto Image necesario y le asigna a la propiedad src el valor de la imagen que saldrá al pasar el ratón sobre la inicial.

La función Rollover_Simple recibe un parámetro que le dice si tiene que cambiar o no la imagen. Como veis, document.IMAGEN se refiere a la imagen cuyo ID es IMAGEN. El nombre del ID lo podeis cambiar tranquilamente. Si le toca cambiar, cambia el atributo src por una, y si no, por la otra.

Para terminar de aclarar esto, vamos a ver un ejemplo en funcionamiento cuyo código se facilita tras la imagen.

Código

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
<!--
  function Rollover_Simple(estado)  {
    if(!estado)  document.Una_Imagen.src = "program_off.gif";
    else         document.Una_Imagen.src = "program_on.gif";
    }

  function Precargar_Imagenes()  {
    Imagen = new Image();
    Imagen.src = "program_on.gif";
    }

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

<BODY onLoad="Precargar_Imagenes();">

<A HREF="#" onMouseOver="Rollover_Simple(true);"
onMouseOut="Rollover_Simple(false);"><IMG SRC="program_off.gif"
NAME="Una_Imagen" BORDER=0></A>

</BODY>
</HTML>
```

Nota

A partir de la versión 4 de Explorer y de la 6 de Netscape, intercambiar imágenes es tan sencillo como hacer esto:

```
<IMG SRC="program_off.gif"
 onMouseOver="this.src='program_on.gif'"
 onMouseOut="this.src='program_off.gif'">
```

Como podemos comprobar:

Ejemplo 2: Validar formularios

Por qué validar

A lo largo del curso se han ido viendo los componentes con los que podemos construir formularios para recoger información de algún tipo, así como sus diversas propiedades y métodos. Ahora vamos a ver cómo validar los datos introducidos en un formulario lo más amplio posible que recoja buena variedad de posibilidades.

¿Por qué la necesidad de validar formularios? Por no recargar al servidor de peticiones: cuando enviamos un formulario, si los datos que hay no son correctos, el servidor nos reenviará la solicitud del formulario pidiéndonos los datos correctos. Con una validación previa nos aseguramos de que el formulario no se envía hasta que ciertos campos estén llenos, otros campos tengan un formato concreto, etc.

Ejemplo base

En este ejemplo, vamos a desarrollar un formulario completo. El código HTML para generar el formulario es el siguiente:

```
<CENTER><B STYLE="font-size:14pt;color:navy;">Foro:
Nuevo usuario</B></CENTER><P>

Regístrate. Introduce bien tus datos y serás dado de alta
automáticamente en el servicio. Es importante que <B>no</B>
escribas caracteres especiales en el nombre de usuario y en la
contraseña: sólo caracteres alfabéticos sin acentuar (la eñe no),
números y el carácter guión bajo _. Si no lo haces así, no serás
dado de alta correctamente. <P>

<CENTER STYLE="background-color:#F5FAFE;">
<FORM ACTION="/cgi-bin/Registro.cgi"
      METHOD="POST" NAME="Registro">
<TABLE BORDER=0 CELLSPACING=2 CELLPADDING=2>
 <TR> <TD>Nombre<SUP>*</SUP>: </TD>
  <TD COLSPAN=2><INPUT TYPE=TEXT SIZE=20 NAME="Nombre"></TD>
 </TR>
 <TR> <TD>Apellidos<SUP>*</SUP>: </TD>
  <TD COLSPAN=2><INPUT TYPE=TEXT SIZE=20 NAME="Apellidos"></TD>
 </TR>
 <TR> <TD>e-mail<SUP>*</SUP>: </TD>
  <TD COLSPAN=2><INPUT TYPE=TEXT SIZE=20 NAME="Email"></TD>
 </TR>
 <TR> <TD>Página web: </TD>
  <TD COLSPAN=2><INPUT TYPE=TEXT SIZE=20 NAME="Web"></TD>
 </TR>
 <TR> <TD>Edad: </TD>
  <TD><INPUT TYPE=TEXT SIZE=3 MAXLENGTH=3 NAME="Edad"></TD>
```

```
  <TD>Sexo:
    <INPUT TYPE=RADIO NAME="Sexo" VALUE="Hombre" CHECKED>Hombre
    <INPUT TYPE=RADIO NAME="Sexo" VALUE="Mujer">Mujer</TD>
 </TR>
</TABLE><P>
Fecha de nacimiento:
 <INPUT TYPE=TEXT SIZE=2 MAXLENGTH=2 NAME="FNDia">
de <SELECT NAME="FNMes">
<OPTION VALUE="">
<OPTION VALUE="1">Enero
<OPTION VALUE="2">Febrero
<OPTION VALUE="3">Marzo
<OPTION VALUE="4">Abril
<OPTION VALUE="5">Mayo
<OPTION VALUE="6">Junio
<OPTION VALUE="7">Julio
<OPTION VALUE="8">Agosto
<OPTION VALUE="9">Septiembre
<OPTION VALUE="10">Octubre
<OPTION VALUE="11">Noviembre
<OPTION VALUE="12">Diciembre
</SELECT> de
<INPUT TYPE=TEXT SIZE=4 MAXLENGTH=4 NAME="FNAnyo"> <BR>
<TABLE BORDER=0 WIDTH="80%" CELLSPACING=2 CELLPADDING=2>
 <TR> <TD>Escoge un nombre de usuario<SUP>*</SUP>:  </TD>
  <TD><INPUT TYPE=TEXT SIZE=15 NAME="Usuario"></TD> </TR>
 <TR> <TD>Introduce tu clave<SUP>*</SUP>:  </TD>
  <TD><INPUT TYPE=PASSWORD SIZE=15 NAME="Clave1"></TD> </TR>
 <TR> <TD>Vuelve a escribir tu clave<SUP>*</SUP>:  </TD>
  <TD><INPUT TYPE=PASSWORD SIZE=15 NAME="Clave2"></TD> </TR>
</TABLE>
<INPUT TYPE="SUBMIT" VALUE="Registrar"
  onClick="return Validar(this.form);">
<INPUT TYPE="RESET" VALUE="Borrar datos">
</FORM>
</CENTER> <P>

<P STYLE="font-family:Verdana,sans-serif;font-size:x-small">
<B>Nota:</B> Los campos marcados con * son
obligatorios, y no serás dado de alta si no los rellenas.
El resto de campos son optativos, y no es necesario que
```

```
los rellenes si no lo deseas. En cualquier caso, los
datos que des son estrictamente confidenciales, y el
único uso que haré de ellos será tener tu ficha de usuario,
pero no serán cedidos ni vendidos a entidad o
persona alguna. Tu nombre de usuario será lo único que se muestre
en los foros a no ser que quieras mostrar más datos, pero eso es
algo que luego podrás configurar a tu elección. </P>
```

Este código nos genera el siguiente formulario, que perfectamente podría servir para registrar a un usuario en un foro (qué haría el CGI del servidor es otro tema):

Foro: Nuevo usuario

Regístrate. Introduce bien tus datos y serás dado de alta automáticamente en el servicio. Es importante que **no** escribas caracteres especiales en el nombre de usuario y en la contraseña: sólo caracteres alfabéticos sin acentuar (la eñe no), números y el carácter guión bajo _. Si no lo haces así, no serás dado de alta correctamente.

Nombre*:
Apellidos*:
e-mail*:
Página web:
Edad: Sexo: Hombre Mujer
Fecha de nacimiento: de de

Escoge un nombre de usuario*:
Introduce tu clave*:
Vuelve a escribir tu clave*:

Registrar Borrar datos

Nota: Los campos marcados con * son obligatorios, y no serás dado de alta si no los rellenas. El resto de campos son optativos, y no es necesario que los rellenes si no lo deseas. En cualquier caso, los datos que des son estrictamente confidenciales, y el único uso que haré de ellos será tener tu ficha de usuario, pero no serán cedidos ni vendidos a entidad o persona alguna. Tu nombre de usuario será lo único que se muestre en los foros a no ser que quieras mostrar más datos, pero eso es algo que luego podrás configurar a tu elección.

Vamos ahora a analizar las necesidades de este formulario. Hemos marcado algunas casillas como introducción obligatoria de datos: esto quiere decir que esos campos no se pueden quedar vacíos. Por tanto, habrá que comprobar que la correspondiente propiedad `value` del objeto `text` sea distinta de la cadena vacía, `""`. Además, la dirección de e-mail debe tener al menos una `@` y no puede ser el primer carácter dentro de la dirección. Podremos comprobar de paso que no existan otros caracteres erróneos en la dirección. Finalmente, las claves no sólo no pueden ser cadena vacía, sino que además deben coincidir. Esta es la primera funcionalidad que debemos implementar.

Pero, ya que nos ponemos, vamos a ser un poco más coherentes. Con la selección que hagan de sexo no vamos a tener más problemas, siempre y cuando mantengamos el hecho de que hay una opción marcada por defecto. La edad no es obligatoria, pero si la introdujeran, estaría bien comprobar que se trata de una edad válida, es decir, que sea un número entero y, además, que sea positivo. La página web tampoco es obligatoria, pero si la escriben, hay que asegurararse de que como mínimo empiece por http://, así como de que hay al menos un punto . Lo mismo cabe decir de la fecha de nacimiento: si no la introducen, se da por válida, pero si se introduce algún campo, entonces se valida (por eso se ofrece un campo en blanco para el mes, para no obligar a ponerlo).

Para permitir una mayor reutilización del código, se ha escrito en un fichero aparte del documento HTML, llamado registro.js, las diversas funciones de validación del formulario. Para incluir este fichero en nuestro documento HTML, escribiremos en la cabecera del mismo:

```
<SCRIPT LANGUAGE="JavaScript" SRC="registro.js">
</SCRIPT>
```

Con esto, ya tenemos disponibles las funciones de validaciones en el documento.

El fuente completo viene documentado para poder adaptar fácilmente estas funciones a vuestras necesidades. Podeis encontrarlo junto con los ficheros que componen el curso. A pesar de ello, reproduzco su contenido a continuación.

El fichero registro.js

Este es el contenido de las funciones de validación del formulario:

```
/* Fichero: registro.js */

/* Funciones para la validacion previa del
   formulario de registro */

/* Copyright de estas funciones: Lola Cárdenas Luque.

   Este fichero puede usarse libremente para validar
   formularios siempre y cuando esté completo. La
   función 'Validar' puede modificarse para que se
   ajuste lo mejor posible a las necesidades */

/* Funciones incluidas:

   Validar(Form) -> Llama a las funciones correspondientes
                    que validan campos de forma separada.
                    El parámetro 'Form' es una referencia
                    al formulario desde el que es llamada.
     Modo de uso:
       onClick='return Validar(this.form);'
       dentro de un formulario

   ValidarNombre(Nombre, Apellidos)
```

```
     ValidarEmail(Email)
     ValidarClave(Usuario, Clave1, Clave2)
     ValidarEdad(Edad)
     ValidarFecha(Form)
     ValidarWeb(Web)

     Para usar estas funciones, debes tener creado un formulario
     cuyos campos reciban nombres muy concretos, o modificar los
     de este script. El campo en el que se escriba el nombre debe
     llamarse 'Nombre', el de los apellidos, 'Apellidos', el del
     e-mail, 'Email', el de la edad, 'Edad'. En la fecha de
     nacimiento, el día debe llamarse 'FNDia', el mes 'FNMes'
     y el año 'FNAnyo'. En cuanto a nombre de usuario y
     contraseña, 'Usuario', 'Clave1' y 'Clave2'. Por último, el
     campo con la dirección de la página web, será 'Web'.

     Es importante respetar estos nombres en los campos si no
     se quiere modificar el script, pues asume que se llamarán
     así. Si se quieren cambiar estos nombres en el código HTML
     del formulario, entonces habrá que cambiar esos valores en
     este fichero.

  */


/* Función genérica para validar el formulario que llama
   a varias funciones: así se desglosa el trabajo y se
   depura más fácilmente en caso de errores */

  function Validar(Form)  {
    return (
         ValidarNombre(Form.Nombre.value, Form.Apellidos.value)
      && ValidarEmail(Form.Email.value)
      && ValidarClave(Form.Usuario.value,
           Form.Clave1.value, Form.Clave2.value)
      && ValidarEdad(Form.Edad.value)
      && ValidarWeb(Form.Web.value)
      && ValidarFecha(Form)
      );
   }
```

```
/* Función para validar los campos con el nombre y apellidos */

  function ValidarNombre(Nombre, Apellidos)  {
    var cadena = "El nombre o el apellido no contiene datos.\n"
      + "No se puede llevar a cabo el registro, revise "
      + "sus datos";

    if ( Nombre == "" || Apellidos == "")  {
      alert(cadena);
      return false;
      }
    else return true;
    }

/* Función para validar la dirección de e-mail */

  function ValidarEmail(email){
    var cadena = "Direccion de correo no valida: " + email
      + "\nPor favor, introduce bien tu direccion";

    if( email.indexOf('@',0) <= 0  || email.indexOf(';',0) != -1
     || email.indexOf(' ',0) != -1 || email.indexOf('/',0) != -1
     || email.indexOf(';',0) != -1 || email.indexOf('<',0) != -1
     || email.indexOf('>',0) != -1 || email.indexOf('*',0) != -1
     || email.indexOf('|',0) != -1 || email.indexOf('`',0) != -1
     || email.indexOf('&',0) != -1 || email.indexOf('$',0) != -1
     || email.indexOf('!',0) != -1 || email.indexOf('"',0) != -1
     || email.indexOf(':',0) != -1 )
       { alert(cadena); return false; }
    else return true;
    }

/* Función para validar el nombre de usuario y la clave */

  function ValidarClave(Usuario, Clave1, Clave2)  {
    var Error0 = "El nombre de usuario no está introducido o "
      + "contiene algún carácter extraño, revíselo";
    var Error1 = "Falta alguno de los campos de la clave "
      + "por rellenar.";
    var Error2 = "Las claves no coinciden.";
```

```
  /* Aquí podeis poner tantos caracteres no permitidos por
     vosotros como querais (dentro del if) */

    if(
        Usuario == "" || Usuario.indexOf('ñ') >= 0
        || Usuario.indexOf('?') >= 0 || Usuario.indexOf('á') >= 0
        || Usuario.indexOf('é') >= 0 || Usuario.indexOf('í') >= 0
        || Usuario.indexOf('ó') >= 0 || Usuario.indexOf('ú') >= 0
       )  {
      alert(Error0);
      return false;
      }

    if( Clave1 == "" || Clave2 == "" ) {
      alert(Error1);
      return false;
      }
    else
      if ( Clave1 != Clave2 ) {
        alert(Error2);
        return false;
        }
      else return true;

    }

/* Función para validar la edad, si es que se
   introduce alguna */

  function ValidarEdad(Edad) {
    var Error = "La edad introducida es inválida.\n"
      + "Revísela, por favor.";

    if (Edad == "") return true;
    else
      if( isNaN(parseInt(Edad)) || parseInt(Edad) <= 0 ) {
        alert(Error);
        return false;
        }
      else return true;
    }
```

```
/* Función para validar la dirección de la página web, si es
   que se introduce alguna */

  function ValidarWeb(Web) {
    var Error = "La dirección web introducida es inválida.\n"
      + "Revísela, por favor.";

    if (Web == "") return true;
    else
      if( Web.substring(0, 7) != 'http://'
          ||
          Web.indexOf('.') < 0
        ) {
        alert(Error);
        return false;
        }
      else return true;
    }


/* Función para validar la fecha, si es que se
   introduce alguna */

  function ValidarFecha(Form)  {
    var Error = "La fecha introducida es inválida.\n"
      + "Revísela, por favor.";
    var Dia, Mes, Anyo, Dia_Mes_Mal = false;
    var MesElegido = Form.FNMes.selectedIndex;

    Dia = Form.FNDia.value;
    Mes = Form.FNMes.options[MesElegido].value;
    Anyo = Form.FNAnyo.value;

    if( Dia == "" && Mes == "" && Anyo == "" ) return true;
    else {
      Dia = parseInt(Dia);
      Mes = parseInt(Mes);
      Anyo = parseInt(Anyo);

      if( !isNaN(Dia) && !isNaN(Mes) && !isNaN(Anyo)
```

```
        &&
        Dia >= 1 && Anyo >= 1900
      )  {

      /* Los meses de Enero, Marzo, Mayo, Julio, Agosto,
         Octubre y Diciembre tienen 31 días */

      if(
         (   Mes == 1 || Mes == 3 || Mes == 5 || Mes == 7
          || Mes == 8 || Mes == 10 || Mes == 12
         )
        && Dia > 31
        )   Dia_Mes_Mal = true;

      /* Los meses de Mayo, Junio, Septiembre
         y Noviembre tienen 30 días */

      if(
         (Mes == 4 || Mes == 6 || Mes == 9 || Mes == 11)
        && Dia > 30
        )   Dia_Mes_Mal = true;

  /* Febrero tiene 28 ó 29 días, dependiendo de si es bisiesto
     o no. Un año es bisiesto si es múltiplo de 4 pero no de
     100 salvo que sea múltiplo de 400. Por ejemplo, 12 y 400
     son bisiestos, pero 100 y 700 no. */

      if( Mes == 2 &&
          ( Dia > 29 ||
            ( Dia == 29 &&
              (
                (Anyo % 400 != 0)
                && (
                   (Anyo % 4 != 0) || (Anyo % 100 == 0)
                   )
              )
            )
          )
        )  Dia_Mes_Mal = true;

      if(Dia_Mes_Mal)  {
```

```
          alert(Error);
          return false;
          }
        else return true;
        }   /* Si la fecha está fuera del rango razonable... */
      else  {
        alert(Error);
        return false;
        }
      }
    }
```

Si alguien quiere ampliarlo con otras sugerencias, que se ponga en contacto conmigo escribiendo a la dirección maddyparadox@wanadoo.es. Igualmente, si alguien no entiende algo del código, puede escribirme para consultar qué es lo que no ve claro.

Ejemplo 3: Quedan N días

De qué se trata

Consiste en un efecto muy sencillo usado para anunciar cuántos días faltan para que suceda un evento. Por ejemplo, si nuestra página web trata sobre nuestro cantante favorito, y resulta que nos hemos enterado de que faltan 20 días para que publique un nuevo disco, querremos anunciarlo en nuestra web, de forma que no tengamos que estar actualizando cada día la página con el tiempo que falte.

Cómo hacerlo

Para poder llevar a cabo este sencillo efecto, debemos manejarnos con el objeto Date. Declararemos una variable que contenga la fecha del evento, y en otra variable almacenaremos la fecha actual. La resta será, en milisegundos, el tiempo que falta para que llegue el evento, así que faltará pasar esos milisegundos a días, para lo que únicamente necesitaremos saber cuántos milisegundos tiene un día. La respuesta es sencilla:

24 x 60 x 60 x 1000 = 86.400.000

Así que ahora vamos a ponernos manos a la obra con el script, que podremos insertar en cualquier punto del documento, dentro de BODY:

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

 var FechaClave, Hoy;
 var FaltanDias;

 FechaClave = new Date(2002, 10, 7); // Por ejemplo [7/11/2002]
 Hoy = new Date();

 FaltanDias = (FechaClave - Hoy)/86400000;

 document.write("<CENTER>Faltan " + Math.round(FaltanDias)
   + " días para el evento :-)</CENTER><P>");

//-->
</SCRIPT>
```

cuyo efecto es:

Faltan -472 días para el evento :-)

Para ajustarlo a nuestras necesidades, simplemente habrá que cambiar el valor de la variable FechaClave, que es la que almacena el valor de la fecha objetivo. Si utilizamos Math.round es para que no aparezcan los decimales de la división efectuada. Si veis este ejemplo después de haber pasado la fecha, entonces los días que saldrán serán negativos. No es incoherente en absoluto: si faltan -3 días para un evento, es porque hace 3 días que sucedió ;-)

Variaciones

Con la misma idea, una fecha clave y la fecha actual, pero invirtiendo el orden de los factores en la resta, tenemos una forma de decir al usuario cuántos días han pasado desde un evento, por ejemplo, cuántos días han pasado desde que se publicó la web :-)

Ejemplo 4: Formateo de páginas

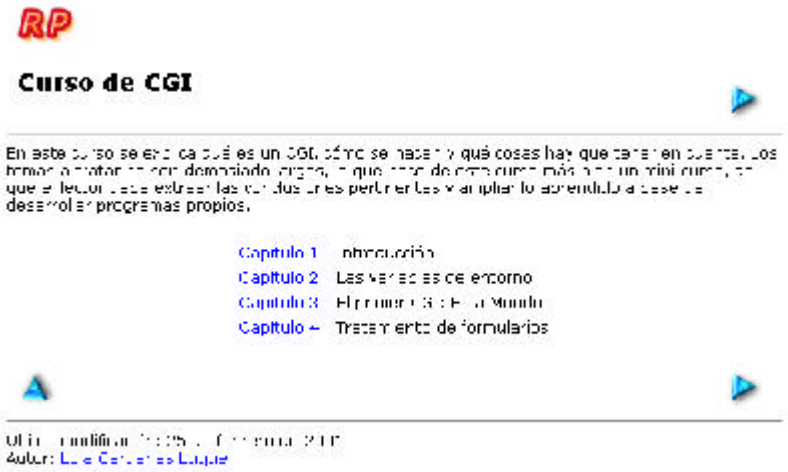
A qué se refiere

Tienes planteado hacer una página web con una estructura muy clara que compartirán la mayoría de sus páginas, y no quieres repetir una y otra vez los mismos comandos HTML para dar el formato adecuado a cada una de las páginas sólo para cambiar un título o los enlaces. Además, resulta que necesitas alguna forma de automatizar la creación de índices, o poder escribir una tabla con novedades que aparece en 50 páginas y no quieres tener que cambiar esas 50 páginas cuando cambien las novedades.

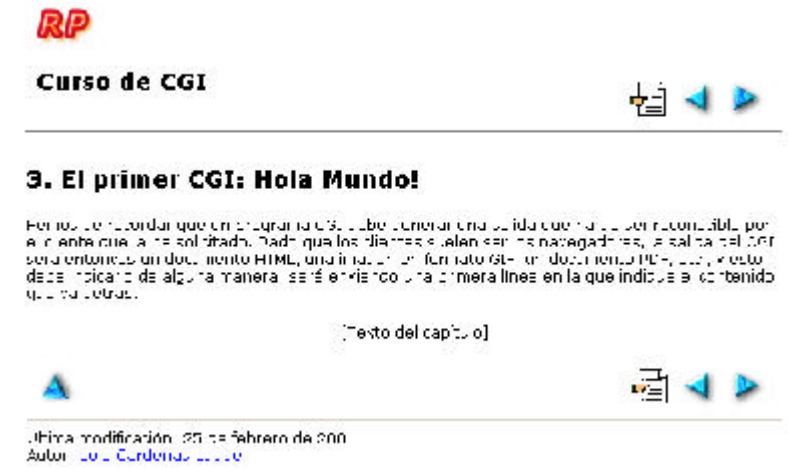
Si tu servidor te ofrece posibilidades como PHP no lo pienses más y encamina tus esfuerzos por esa vía: construirás tus páginas partiendo de unos scripts que te lo dejarán todo como quieres. Pero si tu servidor no te da la opción, siempre queda usar JavaScript para ahorrarnos algo de trabajo.

En este ejemplo, vamos a usar la estructura antigua de los cursos existentes en la web El Rincón del Programador (que es la que conozco, a fin de cuentas ;-)), explicando las funciones empleadas para generar las páginas de los cursos. La estructura nueva es con PHP así que no ha lugar a explicarla en este curso O;-)

Así pues, el resultado buscado será generar automáticamente páginas cuya apariencia sea, por ejemplo:



Índice



Página de un curso

Los estudiaremos separadamente.

Índice

La problemática del índice es la siguiente: en primer lugar, crear una cabecera con el título del curso, un logotipo, una flecha que enlace al primer capítulo, un pie de página con el nombre del autor, su dirección de e-mail, fecha de última actualización y, por supuesto, el propio índice.

Tanto la cabecera como el pie son sencillos: simplemente se crean dos funciones a las que pasar los parámetros oportunos y con ellos escribir el contenido. La fecha de actualización del curso es un poco más engorrosa: si en cada hoja del capítulo escribimos la fecha de actualización, y volvemos a actualizar, tendríamos que cambiar esta fecha en todas las páginas. Como el objetivo no es dar más trabajo sino ahorrárselo, elegiremos esta opción: crear constantes con el nombre del curso que estén en un único fichero, y cuando el curso se actualice, cambiar esta fecha ahí, actualizando con ello todas las páginas afectadas. Seguramente no será la mejor solución, simplemente la que adopté en aquella ocasión. Así pues, las funciones que escribían mis cabeceras y pies de página en el índice, situadas en un fichero idx_cursos.js al mismo nivel que el index.html de los cursos, eran estas:

```
/* Función que inserta 'siguiente' tanto en la
   cabecera como en el pie de página */

function Indice_idx(sig)  {
  if( sig != '' )
    document.write("<A HREF='" + sig + "'>"
      + "<IMG SRC='../../img/right.gif' ALT='Pag. siguiente'"
      + "BORDER=0 HSPACE=5 VALIGN=BOTTOM></A>");
  }
```

Simplemente nos abreviaba un paso de la cabecera: en la siguiente función es donde se emplea:

```
/* Función que inserta una cabecera estándar */

function Cabecera_idx(curso, sig)  {
  with(document)  {
    write("<A NAME='top'> </A>");
    write("<CENTER> <TABLE BORDER=0 WIDTH=98%> <TR>");
    write("<TD ALIGN=LEFT> <A HREF='../index.html'>"
      + "<IMG SRC='../../../img/logo.gif' ALT='Logo' "
      + "BORDER=0></A>"
      + "<H2>" + curso + "</H2></TD>");
    write("<TD ALIGN=RIGHT VALIGN=BOTTOM>");
    Indice_idx(sig);
    write("</TD></TR></TABLE></CENTER><HR><DIV>");
    }
  }
```

A esta función le pasamos el título del curso y el nombre del primer capítulo, y entonces ya crea la cabecera con el título, el enlace, la imagen del logotipo y la imagen de enlace al primer capítulo, siguiendo las rutas adecuadas.

Ahora veamos cómo generaba los pies de página:

```
/* Función que inserta un pie de página estándar */
```

```
function Pie_Pagina_idx(autor, email, ID, sig)  {
  var fecha;

  switch(ID)  {
    case "INT_JS"    : fecha = "1 de julio de 2001"; break;
    case "INT_DELPHI": fecha = "24 de junio de 2001"; break;

    /* Más cursos [...] */

    case "PR_GRF_MAT" : fecha = "6 de mayo de 2001"; break;
    case "DELPHI_MISC": fecha = "11 de julio de 2001"; break;
    case "DELPHI_VENT": fecha = "11 de julio de 2001"; break;
    default: break;
    }

  with(document)  {
    write("</DIV><CENTER> <TABLE BORDER=0 "
      + "WIDTH='98%'> <TR>");
    write("<TD ALIGN=LEFT> <A HREF='#top'>"
      + "<IMG SRC='../../img/up.gif' ALT='Arriba' "
      + "BORDER=0></A></TD>");
    write("<TD ALIGN=RIGHT>");
    Indice_idx(sig);
    write("</TD></TR></TABLE></CENTER><HR>");
    write("<DIV>Ultima modificación: " + fecha
      + "<BR>");
    write("Autor: <A HREF='mailto:" + email + "'>"
      + autor + "</A></DIV>");
    }
  }
```

La variable ID es precisamente la "constante" usada para actualizar la fecha de la última revisión del curso cuando fuera preciso, sin necesidad de pasar la fecha en todas las páginas.

Queda por ver cómo resolví en su momento el tema del índice. Simplemente, construí unas funciones que abrían y cerraban una tabla, y en su interior iban sucediéndose los capítulos, para lo que se escribió una función que generaba el enlace correcto con el nombre del capítulo.

```
/* Función que escribe el inicio de la tabla del
   índice de capítulos */

function Inicio_Indice()  {
  document.write("<CENTER><TABLE BORDER=0 CELLSPACING=4>");
```

```
  }

/* Función que escribe el final de la tabla del
   índice de capítulos */

function Fin_Indice()  {
  document.write("</TABLE></CENTER><P>");
  }

/* Función que escribe el capítulo en la tabla del
   índice de capítulos */

function Capitulo(numero,nombre,archivo)  {
  with(document) {
    write("<TR><TD VALIGN=TOP><A HREF='"
      + archivo + "'>Capítulo ");
    write(numero + "</A>  </TD><TD>"
      + nombre + "</TD></TR>");
    }
  }
```

Con todo esto, crear un índice era tan sencillo como incluir el fichero idx_cursos.js y construir un script como este (por ejemplo):

```
<BODY BGCOLOR="white">
<SCRIPT LANGUAGE="JavaScript">
<!--
Cabecera_idx("Curso de XXXXXX", "PrimerCapitulo.html");
//-->
</SCRIPT>

-- Resumen introductorio del curso --

<SCRIPT LANGUAGE="JavaScript">
<!--
  Inicio_Indice();
  Capitulo("1","Lo que sea 1","PrimerCapitulo.html");
  Capitulo("2","Lo que sea 2","SegundoCapitulo.html");
  Capitulo("3","Lo que sea 3","TercerCapitulo.html");
  Capitulo("4","Lo que sea 4","CuartoCapitulo.html");
  Fin_Indice();
```

```
Pie_Pagina_idx("El nombre del autor", "el e-mail", "ID_CURSO",
  "PrimerCapitulo.html");
//-->
</SCRIPT>

</BODY>
```

Habría que comentar la estructura de directorios, pues influye mucho en cómo está hecho el script. Partiendo del raíz, tenía dos directorios, uno con imágenes y otro con los cursos. Dentro del directorio de los cursos, tenía los scripts y un index común. Además, había un directorio por sección. Dentro del directorio de una sección, había un index para la sección y un directorio por curso. Esa era la estructura.

Página de un curso

Construir una página es más sencillo que el índice. En esta ocasión, tendremos las funciones en un archivo llamado cursos.js, al mismo nivel en la estructura de directorios que el comentado, idx_cursos.js. Por qué separar índice de capítulos responde únicamente a razones de tamaño del fichero: tarda menos en cargar un archivo de 6 KB que uno de 12 KB. Así, cada tipo de documento se preocupa únicamente de las funciones que le incumben.

En los cursos vamos a necesitar una cabecera y un pie de página. La idea es similar a la vista en el índice, veamos el código:

```
/* Función que inserta anterior/siguiente tanto en la
   cabecera como en el pie de página */

function Indice(prev, sig)  {
 document.write("<A HREF='index.html'>"
   + "<IMG SRC='../../img/indice.gif' ALT='Indice del curso'"
   + "BORDER=0 HSPACE=5 VALIGN=BOTTOM></A>");

 if( prev != '' )
   document.write("<A HREF='" + prev + "'>"
     + "<IMG SRC='../../img/left.gif' ALT='Pag. anterior'"
     + "BORDER=0 HSPACE=5 VALIGN=BOTTOM></A>");
 if( sig != '' )
   document.write("<A HREF='" + sig + "'>"
     + "<IMG SRC='../../img/right.gif' ALT='Pag. siguiente'"
     + "BORDER=0 HSPACE=5 VALIGN=BOTTOM></A>");
 }

/* Función que inserta una cabecera estándar para los cursos */

function Cabecera(curso, prev, sig)  {
 with(document)  {
```

```
   write("<A NAME='top'> </A>");
   write("<CENTER> <TABLE BORDER=0 WIDTH=98%> <TR>");
   write("<TD ALIGN=LEFT> <A HREF='../index.html'>"
     + "<IMG SRC='../../../img/logo.gif' ALT='Logo' "
     + "BORDER=0></A>"
     + "<H2>" + curso + "</H2></TD>");
   write("<TD ALIGN=RIGHT VALIGN=BOTTOM>");
   Indice(prev, sig);
   write("</TD></TR></TABLE></CENTER><HR><DIV>");
   }
 }

/* Función que inserta un pie de página estándar
   para los cursos */

function Pie_Pagina(autor, email, ID, prev, sig)  {
 var fecha;

 switch(ID)  {
   case "INT_JS"    : fecha = "1 de julio de 2001"; break;
   case "INT_DELPHI": fecha = "24 de junio de 2001"; break;

   /* Más cursos [...] */

   case "PR_GRF_MAT" : fecha = "6 de mayo de 2001"; break;
   case "DELPHI_MISC": fecha = "11 de julio de 2001"; break;
   case "DELPHI_VENT": fecha = "11 de julio de 2001"; break;
   default: break;
   }

 with(document)  {
   write("</DIV><CENTER> "
     + "<TABLE BORDER=0 WIDTH='98%'> <TR>");
   write("<TD ALIGN=LEFT> <A HREF='#top'>"
     + "<IMG SRC='../../img/up.gif' ALT='Arriba' "
     + "BORDER=0></A></TD>");
   write("<TD ALIGN=RIGHT>");
   Indice(prev, sig);
   write("</TD></TR></TABLE></CENTER><HR>");
   write("<DIV>Última modificación: " + fecha + "<BR>");
   write("Autor: <A HREF='mailto:" + email + "'>"
```

```
      + autor + "</A></DIV>");
    }
  }
```

Una posible forma de usarlas es poniendo tanto anterior como siguiente, pero podemos no poner alguno (como es el caso de la primera página o de la última). Así que vamos a ver un ejemplo en el que nos falta "anterior", es decir, cómo usarlo en el caso de la primera página de un curso:

```
<HTML>
<HEAD>
 <SCRIPT LANGUAGE="JavaScript" SRC="../../cursos.js">
 </SCRIPT>
</HEAD>
<BODY BGCOLOR=white>

<SCRIPT LANGUAGE="JavaScript">
<!--
Cabecera("Curso de XXXXXXXX", "", "Capitulo2.html");
//-->
</SCRIPT>

-- Texto del capítulo --

<SCRIPT LANGUAGE="JavaScript">
<!--
Pie_Pagina("Autor", "e-mail", "ID_CURSO", "", "Capitulo2.html");
//-->
</SCRIPT>

</BODY>
</HTML>
```

Ejemplo 5: Abrir ventanas

De qué se trata

Sabemos que desde JavaScript podemos abrir ventanas con unas ciertas características. En este ejemplo simplemente vamos a dejar caer unas funciones que nos permitirán abrir ventanas de características varias.

Ventana cualquiera

Con esta función podremos abrir una ventana genérica: le pasaremos las opciones que queremos en una cadena (con/sin barra de botones, con/sin barra de estado, tamaño reajustable o no, etc), el tamaño y la posición en la que queremos que se abra. También podremos decirle si la queremos centrada, con lo que se ignorarán los parámetros de posición.

```
function AbreVentana(URL_Ventana, OpcionesVentana, Ancho, Alto,
  Centrada, PosX, PosY) {

  if (Centrada) {
    PosX = (screen.availWidth - Ancho)/2;
    PosY = (screen.availHeight - Alto)/2;
    }

  if (OpcionesVentana == '')
    OpcionesVentana = 'width=' + Ancho;
  else
    OpcionesVentana += ',width=' + Ancho;

  OpcionesVentana += ',height=' + Alto + ',left=' + PosX
    + ',top=' + PosY;

  window.open(URL_Ventana, "", OpcionesVentana);
  }
```

Esta función puede usarse de varias formas, por ejemplo así:

```
AbreVentana("MiDirectorio/MiDoc.html", "scrollbars=1",
  400, 500, true);
AbreVentana("MiDirectorio/MiDoc.html", "", 400, 500,
  false, 10, 20);
```

Como vemos, al poner, en la primera, Centrada = true, no es necesario que especifiquemos más: los valores de posición se calculan en la función.

Podemos aprovechar esta función genérica para crear funciones nuevas con características concretas que usemos a menudo, funciones que llamarán convenientemente a la que acabamos de definir.

Ventana de tamaño y posición dados

Usando la función anterior, vamos a crear otra que permitirá abrir ventanas con barra de scroll (lo que significa que el resto de características están deshabilitadas) y el tamaño y posición que queramos, permitiendo centrar:

```
function AbreVentanaSB(URL_Ventana, Ancho, Alto, Centrada,
  PosX, PosY) {

  AbreVentana(URL_Ventana, "scrollbars=1", Ancho, Alto,
    Centrada, PosX, PosY);
  }
```

Por ejemplo, podremos llamar a la función como sigue:

```
AbreVentanaSB("Pagina.html", 200, 100, false, 10, 10);
AbreVentanaSB("Pagina.html", 400, 300, true);
```

Ventana centrada

Usando la primera función, vamos a crear otra que permitirá abrir ventanas con barra de scroll y el tamaño que queramos, apareciendo siempre centrada:

```
function AbreVentanaCSB(URL_Ventana, Ancho, Alto) {
  AbreVentana(URL_Ventana, "scrollbars=1", Ancho, Alto, true);
  }
```

Podremos llamar a esta función así:

```
AbreVentanaCSB("Pagina.html", 400, 200);
```

El objeto screen

Para poder centrar la ventana, de alguna manera necesitamos conocer la resolución de pantalla de quien está navegando. Para eso tenemos el objeto screen.

No hemos hablado de él en el curso por tratarse de una incorporación posterior, pero sí que mencionaremos algunos detalles: contiene información sobre la pantalla de quien está navegando, y entre otras cosas, nos puede informar de la resolución del cliente. Las propiedades availWidth y availHeight nos dicen el ancho y alto (respectivamente) disponibles en pantalla, mientras que las propiedades width y height nos dicen el total. Como el total incluye la barra de tareas del sistema operativo y algunos otros elementos, hemos usado las dos primeras para centrar la ventana siempre con respecto a lo que queda disponible.

Ejemplo 6: Generador de tablas

En qué consiste

Vamos a desarrollar un script sencillo que tomará datos de un formulario y nos generará el código HTML necesario para crear una tabla con los datos que le hemos dicho. Se trata de un ejemplo para profundizar en la manipulación de formularios.

Cómo hacerlo

En primer lugar, decidimos qué datos vamos a solicitar, y si daremos alguna funcionalidad extra. En este caso, vamos a preguntar el ancho de la tabla y daremos a elegir si es en pixels o en porcentaje. Así que manos a la obra, y creemos el formulario siguiente:

```
<FORM NAME="GeneraTabla">
<CENTER><TABLE BORDER=0>
<TR>
 <TD>Introduce el número de filas:</TD>
 <TD><INPUT TYPE=TEXT SIZE=5 NAME="NumFilas"></TD>
</TR>
<TR>
 <TD>Introduce el número de columnas:</TD>
 <TD><INPUT TYPE=TEXT SIZE=5 NAME="NumColumnas"></TD>
</TR>
<TR>
 <TD>Introduce el ancho de la tabla:</TD>
 <TD><INPUT TYPE=TEXT SIZE=5 NAME="Ancho"></TD>
</TR>
<TR>
 <TD COLSPAN=2>
 El ancho es en:
 <INPUT TYPE=RADIO NAME="AnchoTabla" VALUE="0" CHECKED> Pixels
 <INPUT TYPE=RADIO NAME="AnchoTabla" VALUE="1"> Porcentaje
 </TD>
</TR>
<TR>
 <TD>Introduce el borde de la tabla:</TD>
 <TD><INPUT TYPE=TEXT SIZE=5 NAME="Borde"></TD>
</TR>
</TABLE></CENTER><P>

<CENTER>
<INPUT TYPE="BUTTON" VALUE="Crear la tabla"
  onClick="CrearTabla(this.form)";>
<INPUT TYPE=RESET VALUE="Borrar tabla"></CENTER><P>
<B>Tabla creada:</B> <P>
<CENTER>
```

```
<TEXTAREA ROWS=10 COLS=40 NAME="Texto"></TEXTAREA>
</CENTER>
</FORM>
```

que tiene este aspecto:

Introduce el número de filas:

Introduce el número de columnas:

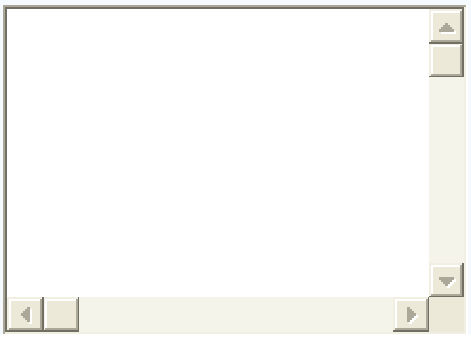
Introduce el ancho de la tabla:

El ancho es en: Pixels Porcentaje

Introduce el borde de la tabla:

Borrar tabla

Tabla creada:



Ahora falta ver cómo es la función CrearTabla, que será quien tome los parámetros entrados, los trate y cree el código HTML para generar la tabla:

```
function CrearTabla(Form) {
  var i, j;
  var NFilas, NColumnas, ancho, anchoTabla, borde, texto;

  anchoTabla = '';

  with(Form) {
    NFilas = (
                NumFilas.value == ''
             || NumFilas.value.indexOf(' ') >= 0
             || isNaN(NumFilas.value)
            ) ? 1 : parseInt(NumFilas.value);
    NColumnas = (
                   NumColumnas.value == ''
                || NumColumnas.value.indexOf(' ') >= 0
                || isNaN(NumColumnas.value)
```

```
                ) ? 1 : parseInt(NumColumnas.value);
    ancho = (
                Ancho.value == ''
             || Ancho.value.indexOf(' ') >= 0
             || isNaN(Ancho.value)
            ) ? 1 : parseInt(Ancho.value);
    borde = (
                Borde.value == ''
             || Borde.value.indexOf(' ') >= 0
             || isNaN(Borde.value)
            ) ? 1 : parseInt(Borde.value);
    }

  if(Form.AnchoTabla[1].checked) anchoTabla = '%';

  texto = '<TABLE BORDER="' + borde
    + '" WIDTH="' + ancho + anchoTabla + '">\n';

  for(i = 1; i <= NFilas; i++) {
    texto += '<TR>\n';

    for(j = 1; j <= NColumnas; j++)
      texto += '<TD> </TD>\n';

    texto += '</TR>\n';
    }

  texto += '</TABLE>';

  Form.Texto.value = texto;
  }
```

Como se ve, la idea es muy sencilla: comprobamos que cada campo que ha de ser numérico no esté vacío, tenga espacios o no sea un número. Si pasa alguna de estas tres condiciones, tomamos como valor por defecto 1 (por poner alguno), pero si no, cogemos el valor introducido por el usuario. Teniendo ya números válidos, simplemente recorremos el número de filas y de columnas para crear la tabla, asignando previamente el texto a insertar a una variable auxiliar intermedia, texto sobre la que se construye el código a generar, y que al finalizar asignamos a la propiedad value del campo textarea, lo que implica automáticamente que se escribe en su área y lo tenemos disponible para copiar y pegar.

Ejemplo 7: Marcar todas las CheckBoxes

En qué consiste

En este ejemplo explicamos cómo marcar todas las casillas de tipo CheckBox que haya en un formulario dentro de una página. Para ello, todas las casillas deben tomar el mismo valor en el parámetro NAME.

Cómo conseguirlo

Haremos un formulario de ejemplo, cuyo código es:

```
<FORM NAME="Ejemplo">
Elige colores: <P>
 <INPUT TYPE=CHECKBOX NAME="ChkBox1" VALUE="Azul">Azul<BR>
 <INPUT TYPE=CHECKBOX NAME="ChkBox1" VALUE="Rojo">Rojo<BR>
 <INPUT TYPE=CHECKBOX NAME="ChkBox1" VALUE="Verde">Verde<BR>
 <INPUT TYPE=CHECKBOX NAME="ChkBox1" VALUE="Blanco">Blanco<BR>
 <INPUT TYPE=CHECKBOX NAME="ChkBox1" VALUE="Negro">Negro<P>
 <INPUT TYPE=BUTTON VALUE="Elegir todos los colores"
  onClick="this.value = ElegirTodos(this.form.ChkBox1, 0);"><P>

Elige sabores: <P>
 <INPUT TYPE=CHECKBOX NAME="ChkBox2" VALUE="Chocolate">
 Chocolate<BR>
 <INPUT TYPE=CHECKBOX NAME="ChkBox2" VALUE="Naranja">Naranja<BR>
 <INPUT TYPE=CHECKBOX NAME="ChkBox2" VALUE="Tomate">Tomate<BR>
 <INPUT TYPE=CHECKBOX NAME="ChkBox2" VALUE="Melon">Melón<P>
 <INPUT TYPE=BUTTON VALUE="Elegir todos los sabores"
  onClick="this.value = ElegirTodos(this.form.ChkBox2, 1);">
</FORM>
```

que tiene la siguiente apariencia:

Elige colores:

☐ Azul
☐ Rojo
☐ Verde
☐ Blanco
☐ Negro

Elige sabores:

☐ Chocolate
☐ Naranja

 Tomate

 Melón

En la cabecera del documento HTML tenemos este código:

```
<SCRIPT LANGUAGE="JavaScript">
<!--
var TodosChecked = new Array(false, false);
var TxtMarcar = new Array('Elegir todos los colores',
  'Elegir todos los sabores');
var TxtDesmarcar = new Array('Deseleccionar todos los colores',
  'Deseleccionar todos los sabores');

function ElegirTodos(ChkBox, QueCheck) {
  var i;

  for(i = 0; i < ChkBox.length; i++)
    ChkBox[i].checked = !TodosChecked[QueCheck];

  TodosChecked[QueCheck] = !TodosChecked[QueCheck];

  return (TodosChecked[QueCheck] ? TxtDesmarcar[QueCheck] :
                                   TxtMarcar[QueCheck]);
  }
//-->
</SCRIPT>
```

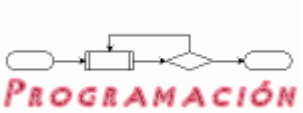
Como tenemos dos CheckBoxes distintas, el texto que aparece en el botón para Marcar/Desmarcar en principio es distinto. Aunque fuera igual, hay un detalle más importante. Si mantenemos una única variable TodosChecked y no un array con tantos valores como grupos de CheckBoxes, veamos un ejemplo de lo que pasaría:

Inicialmente, tenemos los dos grupos desmarcados. Pulsamos el botón de uno de ellos. Entonces el valor TodosChecked se pondría a true. Si ahora pulsamos el botón del otro grupo, como TodosChecked = true, lo desmarcaría, cuando lo que queremos es marcarlo. Por eso debemos mantener los estados independientemente, y después en la función seleccionamos cuál de ellos queremos.

Ejemplo 8: Enmarcar imagen

En qué consiste

Se trata de un simple y a la vez vistoso ejemplo que enmarca una imagen al pasar sobre ella, como podemos ver si pasamos el ratón sobre esta imagen:



En qué consiste

Para que esto funcione, hemos de combinar dos elementos: un estilo CSS y una función que sea capaz de cambiar ese estilo.

El estilo CSS será el encargado de dar el borde a la imagen: diseñaremos nuestra página teniendo en cuenta que el color de fondo tendrá que ser sólido, pues este mismo color será el que asignemos al estilo. Por ejemplo, como esta página tiene el fondo de color blanco, el estilo definido en la cabecera del documento es:

```
<STYLE TYPE="text/css">
<!--

.ImagenConMarco {
  border:1px solid white;
  }

//-->
</STYLE>
```

Se trata de un marco sólido con 1 pixel de grosor y color blanco. Ahora hay que escribir la función JavaScript que sea capaz de cambiar el color de este marco. Por ejemplo:

```
function CambiarColorBorde(Imagen, Color) {
  if( document.all || document.getElementById )
    Imagen.style.borderColor = Color;
  }
```

Mandamos la referencia de la imagen y el color que le queremos poner al marco. Con document.all en la condición sabemos que el navegador es Explorer 4 o superior (contempla el objeto all, propio suyo, que recoge todos los elementos que cuelgan de document en una página), y con document.getElementById en la condición sabemos que el navegador es Netscape (contempla el método getElementById, propio suyo, que devuelve la referencia de un objeto vía su ID). Estos dos navegadores serán capaces de procesar este efecto. Con style accederemos a los estilos del objeto pasado (la imagen) y cambiamos el color del borde modificando el valor de la propiedad borderColor.

Ahora sólo nos falta ver cómo usar estas dos cosas (el estilo y la función) para crear el efecto visto. Simplemente, habrá que incluir una imagen en el documento con la clase CSS creada para el marco, y con los manejadores de eventos correspondientes:

```
<IMG SRC="IMAGEN.GIF" CLASS="ImagenConMarco"
 onMouseOver="CambiarColorBorde(this, 'maroon');"
```

```
  onMouseOut="CambiarColorBorde(this, 'white');">
```

Añadimos el evento onMouseOut para restaurar el estado inicial de la imagen, es decir sin marco. Realmente, con un marco del mismo color que el fondo de la página, pero como no se nota, nadie se dará cuenta ;-)

Este efecto no es sólo aplicable a imágenes: podemos aplicarlo a cualquier elemento que queramos, siempre y cuando se trate de un elemento de bloque, ya que el marco es una propiedad CSS de bloque.

Aquí tenemos a un párrafo que se ha prestado voluntario para ser enmarcado como si fuera una imagen. Como sabe que se trata de un bloque, se aprovecha, pues un simple SPAN no puede disfrutar de este efecto.

Ejemplo 9: Generador de calendarios

En qué consiste

Con este ejemplo, vamos a crear un ejemplo que nos genere, en una nueva ventana, un calendario del mes del año que le especifiquemos en un formulario. Permitiremos personalizar algunos puntos, como el tamaño de la fuente y los colores, para que el usuario ajuste el diseño del calendario como más le guste.

Formulario de petición de datos

Comenzaremos creando el formulario en el que pediremos los datos al usuario:

```
<FORM NAME="Calendario">


<TABLE BORDER=0>
<TR>
 <TD>Elija el mes:</TD>
 <TD><SELECT NAME="Mes">
 <OPTION VALUE="">
 <OPTION VALUE="1">Enero
 <OPTION VALUE="2">Febrero
 <OPTION VALUE="3">Marzo
 <OPTION VALUE="4">Abril
 <OPTION VALUE="5">Mayo
 <OPTION VALUE="6">Junio
 <OPTION VALUE="7">Julio
 <OPTION VALUE="8">Agosto
 <OPTION VALUE="9">Septiembre
 <OPTION VALUE="10">Octubre
 <OPTION VALUE="11">Noviembre
 <OPTION VALUE="12">Diciembre
 </SELECT></TD>
</TR>
<TR>
 <TD>Elija el año:</TD>
 <TD><INPUT TYPE=TEXT SIZE=5 NAME="Anyo"></TD>
</TR>
<TR>
 <TD>Color para la fuente del título:</TD>
 <TD><INPUT TYPE=TEXT SIZE=8 NAME="ColorTitulo"></TD>
</TR>
<TR>
 <TD>Color para el fondo del título:</TD>
 <TD><INPUT TYPE=TEXT SIZE=8 NAME="ColorFondoTitulo"></TD>
</TR>
<TR>
```

```
 <TD>Tamaño para la fuente del título:</TD>
 <TD><INPUT TYPE=TEXT SIZE=3 NAME="TamTitulo"></TD>
</TR>
<TR>
 <TD>Color para la fuente del calendario:</TD>
 <TD><INPUT TYPE=TEXT SIZE=8 NAME="ColorCalendario"></TD>
</TR>
<TR>
 <TD>Color para el fondo del calendario:</TD>
 <TD><INPUT TYPE=TEXT SIZE=8 NAME="ColorFondoCalendario"></TD>
</TR>
<TR>
 <TD>Tamaño para la fuente del calendario:</TD>
 <TD><INPUT TYPE=TEXT SIZE=3 NAME="TamCalendario"></TD>
</TR>
</TABLE><P>

<INPUT TYPE=BUTTON VALUE="Crear calendario"
 onClick="GenerarCalendario(this.form);">
<INPUT TYPE=RESET VALUE="Borrar datos">
</FORM>
```

que presenta el siguiente aspecto:

Elija el mes:
Elija el año:
Color para la fuente del título:
Color para el fondo del título:
Tamaño para la fuente del título:
Color para la fuente del calendario:
Color para el fondo del calendario:
Tamaño para la fuente del calendario:

Borrar datos

Trataremos de validar los datos de alguna manera para que no salgan cosas raras en el calendario. Los colores tendrán que seguir el formato estándar para HTML, esto es, una cadena con el formato #RRGGBB, donde RR es la componente roja del color, GG la verde y BB la azul, en hexadecimal todas ellas. Si no se escribe un color de acuerdo a este formato, no se dará por válido el formulario. Igualmente, si el tamaño de la fuente resulta ser un valor incorrecto. El formulario tampoco será válido si faltan el mes o el año, o si el año es incorrecto.

Para validar el formulario y crear el calendario, tenemos definidas las siguientes funciones en la cabecera del documento:

```
function EsHexadecimal(cadena, cuantosCar) {
  var i = 0, EsHex = true;

  while( (i < cadena.length) && EsHex ) {
    EsHex = (   parseInt(cadena.charAt(i)) >= 0
             || parseInt(cadena.charAt(i)) <= 9
             || cadena.charAt(i).toUpperCase() == 'A'
             || cadena.charAt(i).toUpperCase() == 'B'
             || cadena.charAt(i).toUpperCase() == 'C'
             || cadena.charAt(i).toUpperCase() == 'D'
             || cadena.charAt(i).toUpperCase() == 'E'
             || cadena.charAt(i).toUpperCase() == 'F'
            );
    i++;
    }

  return EsHex;
  }
```

Se trata de una sencilla función que nos dice si una cadena de cuantosCar caracteres corresponde a un número hexadecimal. La usaremos para validar los colores.

La función que valida los datos del formulario es esta:

```
function Validar(Form) {
  return (
     Form.Mes.selectedIndex > 0
  && parseInt(Form.Anyo.value) >= 1900
  && Form.ColorTitulo.value.charAt(0) == '#'
  && Form.ColorTitulo.value.length <= 7
  && EsHexadecimal(Form.ColorTitulo.value.substring(1, 7), 6)
  && Form.ColorFondoTitulo.value.charAt(0) == '#'
  && Form.ColorFondoTitulo.value.length <= 7
  && EsHexadecimal(
       Form.ColorFondoTitulo.value.substring(1, 7), 6)
  && parseInt(Form.TamTitulo.value) > 0
  && Form.ColorCalendario.value.charAt(0) == '#'
  && Form.ColorCalendario.value.length <= 7
  && EsHexadecimal(
       Form.ColorCalendario.value.substring(1, 7), 6)
```

```
  && Form.ColorFondoCalendario.value.charAt(0) == '#'
  && Form.ColorFondoCalendario.value.length <= 7
  && EsHexadecimal(
       Form.ColorFondoCalendario.value.substring(1, 7), 6)
  && parseInt(Form.TamCalendario.value) > 0
         );
  }
```

Para cada uno de los colores comprobamos tres cosas: que el primer carácter (el cero) sea # (como corresponde a un código válido), que la longitud de la cadena sea menor o igual que 7 (1 de # más 6 del código RRGGBB) y que los caracteres 1 a 6 sean una cadena hexadecimal válida.

Por último, la función que crea el calendario (que es la llamada por el botón del formulario) resulta:

```
function GenerarCalendario(Form) {
  var posX, posY, ancho = 600, alto = 400;
  var Ventana;  // Referencia a la ventana con el calendario

  var i, j, NumFilas, aux = 1;

  var FechaInicial, CuantosDias, DiasMes, DiaInicial;
  var MesElegido;

  var DiasSemana = new Array('Lunes', 'Martes',
                   'Miércoles', 'Jueves', 'Viernes',
                   'Sábado', 'Domingo');

  var Meses = new Array('Enero', 'Febrero', 'Marzo',
              'Abril', 'Mayo', 'Junio', 'Julio',
              'Agosto', 'Septiembre', 'Octubre',
              'Noviembre', 'Diciembre');

  if (!Validar(Form)) {
    alert('Alguno de los datos introducidos es incorrecto\n'
      + 'o faltan datos por rellenar');
    return 0;
    } // Salimos sin crearlo

  posX = (screen.availWidth - ancho)/2;
  posY = (screen.availHeight - alto)/2;

  MesElegido = Form.Mes.selectedIndex - 1;
```

```
FechaInicial = new Date(parseInt(Form.Anyo.value),
                MesElegido, 1);
DiaInicial = (FechaInicial.getDay() == 0) ?
              6 : FechaInicial.getDay() - 1;

/* Abrimos la ventana en la que escribiremos el calendario */

Ventana = open('ejemCalendario.html','',
              'scrollbars=1,left=' + posX
            + ',top=' + posY
            + ',width=' + ancho
            + ',height=' + alto);

with(Ventana.document) {
  /* Escribimos la cabecera con el mes y el año */

  write('<CENTER><TABLE BORDER=0 BGCOLOR='
    + Form.ColorFondoCalendario.value
    + ' CELLSPACING=5>'
    + '<TR><TD COLSPAN=7><TABLE BORDER=0 BGCOLOR='
    + Form.ColorFondoTitulo.value
    + ' WIDTH="100%"><TR><TD STYLE="color:'
    + Form.ColorTitulo.value + ';'
    + 'font-size:' + Form.TamTitulo.value + 'pt;'
    + 'font-weight:bold"'
    + '> ' + Meses[MesElegido] + ' de '
    + Form.Anyo.value + '</TD></TR></TABLE>'
    + '</TD></TR>');

  write('<TR>');

  /* Escribimos los días de la semana */

  for(i = 0; i < DiasSemana.length; i++)
    write('<TD ALIGN=RIGHT><B STYLE=color:'
      + Form.ColorCalendario.value + ';'
      + 'font-size:' + Form.TamCalendario.value
      + 'pt;">'
      + DiasSemana[i] + '</B></TD>');

  write('</TR>');
```

```
/* Escribimos los huecos del calendario */

write('<TR>');
for(i = 0; i < DiaInicial; i++)
  write('<TD></TD>');

DiasMes = DiasDelMes(MesElegido + 1,
  parseInt(Form.Anyo.value));
CuantosDias = DiasMes + i - 1; // Para contar bien las filas

NumFilas = parseInt(CuantosDias/DiasSemana.length);

/* Completamos la primera fila */

for(; i < DiasSemana.length; i++)
  write('<TD ALIGN=RIGHT STYLE=color:'
    + Form.ColorCalendario.value + ';'
    + 'font-size:' + Form.TamCalendario.value
    + 'pt;">' + (aux++)
    + '</TD>');
write('</TR>');

/* Escribimos el resto de las filas */

for(i = 1; i <= NumFilas; i++) {
  write('<TR>');
  j = 0;

  while(   (j < DiasSemana.length)
        && (aux <= DiasMes)
       ) {
    write('<TD ALIGN=RIGHT STYLE=color:'
      + Form.ColorCalendario.value + ';'
      + 'font-size:' + Form.TamCalendario.value
      + 'pt;">' + (aux++)
      + '</TD>');

    j++;
    }
```

```
      /* Cerramos posibles huecos */

      if(j < DiasSemana.length - 1)
        for(; j < DiasSemana.length; j++)
          write('<TD></TD>');

      write('</TR>');
      }

    write('</TABLE></CENTER>');
    }
  }
```

Esta función hace bastantes cosas. En primer lugar, declaramos las variables que vamos a necesitar: una referencia a la ventana que abriremos para escribir el calendario, la posición y tamaño que ocupará, variables para calcular cuántas filas hay que usar en el calendario así como saber qué día de la semana empieza el mes, y los nombres de los días y meses en arrays para escribirlos más fácilmente en bucles.

Después, validamos el formulario. Si hay algún campo incorrecto, salimos de la función sin crear el calendario, en caso contrario (i.e., todos los campos son correctos) procedemos a generarlo. Lo centramos en pantalla y obtenemos la referencia a la ventana creada en la variable Ventana.

Obtenemos el mes elegido, obtenemos el día de la semana en el que empezó ese mes, y procedemos a rellenar la tabla del calendario. Escribimos blancos hasta que nos encontramos con el día de la semana en que empieza el mes, completamos esa fila, y ya seguimos rellenando la tabla hasta que no nos queden días del mes.

Para obtener el número de días que corresponden al mes seleccionado, se emplea la siguiente función auxiliar:

```
function DiasDelMes(Mes, Anyo) {
  if( Mes <= 0 || Mes >= 13) return 0;

  if(   Mes == 1
     || Mes == 3
     || Mes == 5
     || Mes == 7
     || Mes == 8
     || Mes == 10
     || Mes == 12
    ) return 31;

  if(   Mes == 4
     || Mes == 6
     || Mes == 9
     || Mes == 11
```

```
    ) return 30;

/* Aquí llegamos si Mes == 2, i.e., Febrero */

if( (Anyo % 400 == 0)
    || (
       (Anyo % 4 == 0) && (Anyo % 100 != 0)
       )
  ) return 29;

return 28;  // Mes == 2 y año no bisiesto
}
```

Y con eso ya lo tenemos todo listo para crear nuestro formulario.

Ejemplo 10: Arrastrar y soltar

En qué consiste

Es el conocido efecto de mostrar una capa en la pantalla que debe realizar alguna función específica, permitiendo moverla por la pantalla para colocarla donde se desee. Sólo vamos a ver su implementación para Explorer.

Cómo hacerlo

En primer lugar, tenemos que definir una capa vía un estilo que incluya position:absolute que contenga a la imagen. Por ejemplo:

```
<STYLE TYPE="text/css">
 #CapaMovil {
    position:absolute;
    }
</STYLE>
```

Tenemos definido el estilo, ahora definimos la capa propiamente dicha:

```
<DIV ID="CapaMovil" onMouseDown="InicioArrastre();"
 STYLE="left:100px;top:100px;">
 <IMG SRC="GatitoLapiz.gif">
</DIV>
```

Asignar ID=CapaMovil al bloque DIV consigue, por una parte, aplicar el estilo definido al bloque y, por otra parte, asociarle un identificador único para poder manipularlo desde JavaScript. Ahora vamos a ver el código necesario para implementar el efecto. Para ello, tenemos que crear código para los eventos de ratón siguientes: onMouseDown, onMouseUp y onMouseMove.

En primer lugar, asignamos como manejador de los eventos onMouseDown y onMouseUp a las funciones InicioArrastre y FinArrastre, respectivamente.

Ahora, cuando el usuario pulsado el botón izquierdo del ratón, dentro de la función InicioArrastre se captura el evento onMouseMove y se asocia a la función Arrastrar para que sea su manejadora. Además, ponemos a true la variable RatonPulsado para decirnos que está pulsado el botón izquierdo del ratón y que, por tanto, hemos de mover la imagen. Cuando el usuario suelta el botón, se libera la captura del evento onMouseMove, con lo que el arrastre se da por finalizado.

El código completo del ejemplo es este:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<HTML>
<HEAD>
 <STYLE TYPE="text/css">
 <!--
  #CapaMovil {
     position:absolute;
```

```
    }

//-->
</STYLE>
<SCRIPT LANGUAGE="JavaScript">
<!--
 var IE_X, IE_Y, AntX, AntY, RatonPulsado = false;

 function InicioArrastre(Capa) {
   if (event.button == 1) {
     /* Coordenadas auxiliares */

     AntX = CapaMovil.style.pixelLeft;
     AntY = CapaMovil.style.pixelTop;
     IE_X = event.clientX;
     IE_Y = event.clientY;

     RatonPulsado = true;
     document.onmousemove = Arrastrar;
     return false;
     }
   }

 function FinArrastre() {
   if (event.button == 1) {
     RatonPulsado = false;
     return false;
     }
   }

 function Arrastrar() {
   if (RatonPulsado) {
     document.all.CapaMovil.style.posLeft = AntX
       + event.clientX - IE_X;
     document.all.CapaMovil.style.posTop = AntY
       + event.clientY - IE_Y;
     return false;
     }
   }

document.onmouseup = FinArrastre;
```

```
 //-->
 </SCRIPT>

</HEAD>

<BODY BGCOLOR=white>

<DIV ID="CapaMovil" onMouseDown="InicioArrastre();"
 STYLE="left:100px;top:100px;">
 <IMG SRC="GatitoLapiz.gif">
</DIV>

</BODY>
</HTML>
```

Ejemplo 11: Menús desplegables

Este último ejemplo del curso entra un poco más en el terreno del HTML dinámico. Vamos a desarrollar unos sencillos menús que podremos usar en nuestras aplicaciones. Este ejemplo es muy simple y no da la opción de hacer submenús.

NOTA: Este documento sigue el esquema existente en Taller Web de la página HTML en castellano. De hecho, los menús han sido desarrollados siguiendo sus indicaciones, aunque cambiando algún pequeño punto.

De qué se trata

El efecto va a consistir en crear unos menús, al estilo de los menús usuales de los programas gráficos, que estén ocultos y que aparezcan cuando el ratón pase sobre la zona del documento definida para que se abra el menú. Además, cuando se abra un menú veremos si existe otro menú abierto para, en ese caso, cerrarlo antes de abrir el nuevo.

Cómo llevarlo a cabo

Lo primero que tenemos que hacer es definir una capa CSS con el estilo que queremos que tenga el menú, si es que queremos que tenga algún estilo especial, y decir que esta capa va a tener posición absoluta. Llamaremos a la capa Menu porque será el estilo que apliquemos a los menús:

```
.Menu {
  position:absolute;
  visibility:hidden;
  background-color:##DFE3F2;  // Para Explorer 4+, Netscape 6
  layer-background-color:##DFE3F2; // Para Netscape 4
  color:navy;
  border:1px solid black;
  padding:5px;
  font-family:Verdana,Arial,Helvetica,sans-serif;
  font-size:10pt;
  }

.Menu A:hover{
  text-decoration:underline;
  font-weight:bold;
  color;navy
  }

.Menu A {
  text-decoration:none;
  color:navy
  }
```

Hemos definido el que será el estilo de la capa de los menús, así como el estilo de los enlaces que tendrán estos menús. Con estos estilos, podemos ponernos a definir las capas propiamente dichas que harán de menús. Por ejemplo:

```
<DIV ID="Menu0" CLASS="Menu">
 <A HREF="../JavaScript2a.html">Sintaxis I: Variables.
 Constantes</A><BR>
 <A HREF="../JavaScript2b.html">Sintaxis I: Tipos de
 datos</A>
</DIV>

<DIV ID="Menu1" CLASS="Menu">
 <A HREF="../JavaScript3a.html">Operadores:
 aritméticos</A><BR>
 <A HREF="../JavaScript3b.html">Operadores:
 Comparación</A><BR>
 <A HREF="../JavaScript3c.html">Operadores:
 Lógicos</A><BR>
 <A HREF="../JavaScript3d.html">Operadores:
 A nivel de bit</A><BR>
 <A HREF="../JavaScript3e.html">Operadores:
 Resto</A>
</DIV>
```

Como son capas cuyo estilo es la clase Menu, al cargarse el documento en principio no veremos nada. Para que aparezcan, por una parte tendremos que asociar al evento onMouseOver de los enlaces (por ejemplo) un manejador que muestre el menú y oculte los demás. Por otra parte, tenemos que hacer que JS se entere de alguna forma que tenemos unos menús que mostrar/ocultar, y para ello tendremos que crearnos un objeto TMenu con las definiciones necesarias para tratar los menús desde JS, y un array de objetos de tipo TMenu que serán ya los menús que trataremos en la página. Además, tendremos un objeto de tipo TMenu que no formará parte del array, que nos servirá para tener la referencia del menú activo y así saber qué menú ocultar cuando vayamos a mostrar otro:

```
var Menu = new Array();
var MenuActivo = null;  // Inicialmente no hay menús activos
```

Donde la definición del objeto TMenu resulta:

```
function TMenu(IdCapa, PosX, PosY) {
  this.Activar = ActivarTMenu;
  this.Mostrar = MostrarTMenu;
  this.Ocultar = OcultarTMenu;

  this.MoverA = MoverTMenuA;

  this.sRefCapa = Navegador.NS4 ? 'document["' + IdCapa + '"]' :
                  'document.all["' + IdCapa + '"]';
```

```
  this.sRefEstilo = Navegador.NS4 ? '' : '.style';
  this.sRefLeft = Navegador.NS4 ? '.left' : '.pixelLeft';
  this.sRefTop = Navegador.NS4 ? '.top' : '.pixelTop';

  this.MoverA(PosX, PosY);
  }
```

Repasando el tema de objetos, vemos que lo que se hace en las primeras cuatro líneas es asignar referencias de los que serán los métodos usados por cualquier objeto de tipo TMenu. Estos métodos aún no están definidos, los veremos después. Por otro lado, en función del navegador, asignamos unos atributos que son cadenas, que contienen el valor de la propiedad válida para el navegador que nos permitirán: referenciar la capa (sRefCapa), modificar el estilo (sRefEstilo) y modificar las posiciones horizontal y vertical (sRefLeft y sRefTop).

El objeto TMenu hace referencia a otro: el objeto TNavegador. Este objeto también será creado por nosotros, con la finalidad de determinar desde el principio con qué navegador están viendo la página para así decidir qué cadenas usaremos para cambiar propiedades de los menús. La definición de dicho objeto será:

```
function TNavegador() {
  this.NS4 = document.layers;
  this.NS6 = document.getElementById;
  this.IE4 = document.all;

  this.DHTML = this.NS4 || this.NS6 || this.IE4;
  }
```

De este objeto crearemos la instancia Navegador como sigue:

```
var Navegador = new TNavegador();
```

Obtenemos el navegador del usuario viendo simplemente qué atributos soporta, pues los atributos elegidos los tiene un navegador pero el otro no.

Gracias a este objeto TNavegador, podremos definir fácilmente los métodos del objeto TMenu:

```
function ActivarTMenu() {
  if (Navegador.DHTML && MenuActivo != this) {
    // Podría ser null, de ahí la comparación
    if (MenuActivo) MenuActivo.Ocultar();

    MenuActivo = this;
    this.Mostrar();
    }
  }
```

```
function MostrarTMenu() {
  eval(this.sRefCapa + this.sRefEstilo
    + '.visibility = "visible"');
  }
function OcultarTMenu() {
  eval(this.sRefCapa + this.sRefEstilo
    + '.visibility = "hidden"');
  }
function MoverTMenuA(x, y) {
  if (Navegador.DHTML) {
    eval(this.sRefCapa + this.sRefEstilo
      + this.sRefTop + ' = y');
    eval(this.sRefCapa + this.sRefEstilo
      + this.sRefLeft + ' = x');
    }
  }
```

Estas funciones crean una cadena con la sentencia a ejecutar: llamando a la función eval conseguimos que dicha sentencia se ejecute. No se comprueba si el navegador soporta o no HTML dinámico porque se asume que únicamente serán llamadas desde ActivarTMenu. Si no fuera así, habría que modificarlas un poco.

Vamos a hacer que aparezcan: para ello, tendremos que tener (por ejemplo) unos enlaces tales que al pasar el ratón sobre ellos, nos desplegarán los menús:

```
<A HREF="JavaScript2.html"
onMouseOver="if(Menu[0]) Menu[0].Activar();">Sintaxis I</A>
<A HREF="JavaScript3.html"
onMouseOver="if(Menu[1]) Menu[1].Mostrar();">Operadores</A>
```

Se incluye la condición por el siguiente motivo: si no se ha terminado de cargar la página, el usuario podría intentar mostrar el menú, dándose la posibilidad de que en ese momento aún no existiera, con lo que se provocaría un error.

Ahora vamos a ver cómo inicializamos los objetos de tipo TMenu que se encargarán de mostrar/ocultar los menús. Para ello, nos crearemos una función que tendrá este aspecto:

```
function InicializarTMenus() {
  if (Navegador.DHTML) {
    if (Navegador.NS4)
      document.captureEvents(Event.MOUSEUP);
    document.onmouseup = OcultarTMenuActivo;
    }

  Menu[0] = new TMenu("Menu0", 40, 30);
```

```
  Menu[1] = new TMenu("Menu1", 150, 30);
  }
```

Comprueba qué navegador está funcionando para realizar la captura de eventos correcta, y a continuación inicializa los objetos de tipo TMenu. La función que oculta el menú activo presenta el siguiente aspecto:

```
function OcultarTMenuActivo(e) {
  if (MenuActivo) {
    MenuActivo.Ocultar();
    MenuActivo = null;
    }
  }
```

Y con esto ya lo tenemos todo. Veamos el código completo del ejemplo:

```
<HTML>
<HEAD>
 <STYLE TYPE="text/css">
 <!--

 .Menu {
   position:absolute;
   visibility:hidden;
   background-color:#DFE3F2;  // Para Explorer 4+, Netscape 6
   layer-background-color:#DFE3F2; // Para Netscape 4
   color:navy;
   border:1px solid black;
   padding:5px;
   font-family:Verdana,Arial,Helvetica,sans-serif;
   font-size:10pt;
   }

 .Menu A:hover{
   text-decoration:underline;
   font-weight:bold;
   color;navy
   }
 .Menu A {
   text-decoration:none;
   color:navy
   }
```

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

/* Objeto TNavegador */

function TNavegador() {
  this.NS4 = document.layers;
  this.NS6 = document.getElementById;
  this.IE4 = document.all;

  this.DHTML = this.NS4 || this.NS6 || this.IE4;
  }

var Navegador = new TNavegador();

var Menu = new Array();
var MenuActivo = null;  // Inicialmente no hay menús activos

/* Métodos de TMenu */
/* TMenu.Activar */

function ActivarTMenu() {
  if (Navegador.DHTML && MenuActivo != this) {
    // Podría ser null, de ahí la comparación
    if (MenuActivo) MenuActivo.Ocultar();

    MenuActivo = this;
    this.Mostrar();
    }
  }

/* TMenu.Mostrar */

function MostrarTMenu() {
  eval(this.sRefCapa + this.sRefEstilo
    + '.visibility = "visible"');
  }
```

```
/* TMenu.OcultarTMenu */

function OcultarTMenu() {
  eval(this.sRefCapa + this.sRefEstilo
    + '.visibility = "hidden"');
  }

/* TMenu.MoverA */

function MoverTMenuA(x, y) {
  if (Navegador.DHTML) {
    eval(this.sRefCapa + this.sRefEstilo
      + this.sRefTop + ' = y');
    eval(this.sRefCapa + this.sRefEstilo
      + this.sRefLeft + ' = x');
    }
  }

/* Objeto TMenu */

function TMenu(IdCapa, PosX, PosY) {
  this.Activar = ActivarTMenu;
  this.Mostrar = MostrarTMenu;
  this.Ocultar = OcultarTMenu;

  this.MoverA = MoverTMenuA;

  this.sRefCapa = Navegador.NS4 ? 'document["' + IdCapa + '"]' :
                  'document.all["' + IdCapa + '"]';
  this.sRefEstilo = Navegador.NS4 ? '' : '.style';
  this.sRefLeft = Navegador.NS4 ? '.left' : '.pixelLeft';
  this.sRefTop = Navegador.NS4 ? '.top' : '.pixelTop';

  this.MoverA(PosX, PosY);
  }



function OcultarTMenuActivo(e) {
  if (MenuActivo) {
    MenuActivo.Ocultar();
    MenuActivo = null;
```

```
    }
  }

 function InicializarTMenus() {
   if (Navegador.DHTML) {
     if (Navegador.NS4)
       document.captureEvents(Event.MOUSEUP);
     document.onmouseup = OcultarTMenuActivo;
     }

   Menu[0] = new TMenu("Menu0", 40, 30);
   Menu[1] = new TMenu("Menu1", 150, 30);
   }

 window.onload = InicializarTMenus;

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

<BODY BGCOLOR=white>

<DIV CLASS="Menu" ID="Menu0">
 <A HREF="../JavaScript2a.html">Sintaxis I:
  Variables. Constantes</A><BR>
 <A HREF="../JavaScript2b.html">Sintaxis I:
  Tipos de datos</A>
</DIV>

<DIV CLASS="Menu" ID="Menu1">
 <A HREF="../JavaScript3a.html">Operadores:
  Aritméticos</A><BR>
 <A HREF="../JavaScript3b.html">Operadores:
  Comparación</A><BR>
 <A HREF="../JavaScript3c.html">Operadores:
  Lógicos</A><BR>
 <A HREF="../JavaScript3d.html">Operadores:
  A nivel de bit</A><BR>
 <A HREF="../JavaScript3e.html">Operadores:
  Resto</A>
</DIV>
```

```
<A HREF="JavaScript2.html"
 onMouseOver="if(Menu[0]) Menu[0].Activar();">Sintaxis I</A>
<A HREF="JavaScript3.html"
 onMouseOver="if(Menu[1]) Menu[1].Activar();">Operadores</A>

</BODY>
</HTML>
```