El lenguaje JavaScript

Asignatura: Programació III
Curso: 1999/2000
Profesor: Toni Navarrete

1. Introducción

Al crear páginas web en HTML nos damos cuenta de que estamos ciertamente restringidos a sólo poder emplear elementos estáticos.

Hay dos soluciones a este problema, con el fin de dotar a nuestras páginas de un mayor dinamismo e interactividad, que pasan por añadir programación o bien en la parte del cliente, o bien en la parte del servidor. Del segundo, programas en el cliente no hablaremos en este curso, pero básicamente, lo que nos permite es conectar nuestra web a una base de datos, de modo que los contenidos de las páginas varíen en función de la información almacenada en la base de datos. En cuanto a añadir programación en la parte del cliente es lo que se trata en este tema: JavaScript.

JavaScript no es más que un sencillo lenguaje de programación, que presenta una característica especial: sus programas, llamados comúnmente scripts, se en las páginas HTML y se ejecutan en el navegador (Netscape Navigator y Microsft Explorer). Estos scripts normalmente consisten en unas funciones que son llamadas desde el propio HTML cuando algún evento sucede. De ese modo, podemos añadir efectos como que un botón cambie de forma al pasar el ratón por encima, o abrir una ventana nueva al pulsar en un enlace, ...

JavaScript fue desarrollado por Netscape, a partir del lenguaje Java, el cual sigue una filosofía similar, aunque va más allá. Java es un lenguaje de programación por sí mismo, como lo puedan ser C, Pascal o VisualBasic. Esto quiere decir, que se puede ejecutar un programa Java fuera de un navegador. Se hablará más sobre Java en posteriores temas. Pero, repetimos, que la diferencia fundamental es que Java es un lenguaje completo, que puede ser utilizado para crear aplicaciones de todo tipo, mientras que JavaScript sólo “funciona” dentro de una página HTML. Por otro lado, también se puede incluir Java en páginas HTML, tal es el caso de los applets, que se podría traducir como “aplicacioncitas”.

2. Scripts y eventos

Para introducir un script JavaScript se hace dentro de las etiquetas:

```
<SCRIPT LANGUAGE=“JavaScript”>
código del script
</SCRIPT>
```

Veamos un ejemplo muy básico de una página HTML que contiene un script de Javascript:

```
<HTML>
<BODY>
<br>
Esto es HTML
<br>
```

```
        <SCRIPT LANGUAGE="JavaScript">
                document.write("Esto es JavaScript")
        </SCRIPT>
<br>
Esto vuelve a ser HTML
</BODY>
</HTML>
```

El resultado es que escribe:

Esto es HTML
Esto es JavaScript
Esto vuelve a ser HTML

Realmente la funcionalidad del anterior script es nula, ya que para escribir "Esto es JavaScript" nos basta con usar HTML. La verdadera utilidad de JavaScript es el que los scripts se ejecuten cuando ocurra algún evento, como sea un click de ratón, que éste pase por encima de un enlace, que se envíen los datos de un formulario, ... En el siguiente ejemplo, se ejecuta el script cuando se pasa por encima del enlace con el ratón:

```
<HTML>
<HEAD>
<BODY>
<a href="#" onmouseover="window.status='este es el texto que aparece en la barra de status'; return true">Pasa el ratón n por aquí</a>
</BODY>
</HTML>
```

En este ejemplo, al pasar el ratón sobre el enlace, se llama a la acción especificada en la cláusula *onmouseover*. Es decir, se ejecuta cuando ocurre un cierto evento. Se dice por eso que es una programación orientada a eventos. En el ejemplo lo que ocurre es que se cambia el texto de la barra de status (la de la parte inferior de la ventana).

Veamos una lista de los eventos más comunes, cuando se producen y sobre qué etiquetas pueden actuar:

Evento	**Se produce cuando ...**	**etiqueta HTML**
onLoad	Al cargar el documento HTML	BODY
onUnload	Al abandonar el doc. HTML	BODY
onMouseOver	Al pasar el ratón por encima del enlace	A
onMouseOut	Al sacar el ratón de encima del enlace	A
onClick	Al clickar, sobre un link o un campo de formulario	A, FORM
onSubmit	Al enviar el formulario	FORM
onFocus	Al activar un campo de edición de un formulario	INPUT
onSelect	Al seleccionar un campo de edición de un formulario	INPUT

onBlur	Al deseleccionar un campo de edición de un formulario	INPUT
onChange	Al cambiar el contenido de un campo de edición o de selección de un formulario	INPUT, SELECT

En posteriores versiones del lenguaje se han ido añadiendo nuevos eventos (aunque a veces presentan problemas de compatibilidad entre Microsoft y Netscape). Ésta es la lista completa de eventos, según la especificación del lenguaje (recordemos que hecha por Netscape):

Event	**Applies to**	**Occurs when**	**Event handler**
`Abort`	images	User aborts the loading of an image (for example by clicking a link or clicking the Stop button)	`onAbort`
`Blur`	windows and all form elements	User removes input focus from window or form element	`onBlur`
`Change`	text fields, textareas, select lists	User changes value of element	`onChange`
`Click`	buttons, radio buttons, checkboxes, submit buttons, reset buttons, links	User clicks form element or link	`onClick`
`DragDrop`	windows	User drops an object onto the browser window, such as dropping a file on the browser window	`onDragDrop`
`Error`	images, windows	The loading of a document or image causes an error	`onError`
`Focus`	windows and all form elements	User gives input focus to window or form element	`onFocus`
`KeyDown`	documents, images, links, text areas	User depresses a key	`onKeyDown`
`KeyPress`	documents, images, links, text areas	User presses or holds down a key	`onKeyPress`
`KeyUp`	documents, images, links, text areas	User releases a key	`onKeyUp`
`Load`	document body	User loads the page in the Navigator	`onLoad`
`MouseDown`	documents, buttons, links	User depresses a mouse button	`onMouseDown`

MouseMove	nothing by default	User moves the cursor	onMouseMove
MouseOut	areas, links	User moves cursor out of a client-side image map or link	onMouseOut
MouseOver	links	User moves cursor over a link	onMouseOver
MouseUp	documents, buttons, links	User releases a mouse button	onMouseUp
Move	windows	User or script moves a window	onMove
Reset	forms	User resets a form (clicks a Reset button)	onReset
Resize	windows	User or script resizes a window	onResize
Select	text fields, textareas	User selects form element's input field	onSelect
Submit	forms	User submits a form	onSubmit
Unload	document body	User exits the page	onUnload

Client-Side JavaScript Guide, Version 1.3. Netscape Communications Corporation, 1999

Continuando con el ejemplo anterior, igualmente se puede hacer que al pulsar un link, sin que cargue directamente una nueva página, se ejecute una función javascript, de la siguiente manera:

```
<a href="javascript:funcion(parametros)">link</a>
```

Esto es muy útil cuando queremos hacer que al pulsar sobre una imagen se cambie la misma, pero que no nos lleve a otra parte. También podría servir para abrir otra ventana, pero sin cambiar el contenido de la antigua.

En un script podemos definir tanto estructuras condicionales como iterativas. Y también podemos definir funciones. Los scripts que contienen estas funciones suelen estar en el head del documento, de manera que se carguen cuando se carga el docuemento. Esas funciones serán llamadas desde el body del documento, cuando ocurra cierto evento. Veamos un ejemplo:

```
<html>
<head>
<script language="LiveScript">
function hola() {
      alert("Hola");
    }
</script>
</head>
<body>
<a href="" onMouseOver="hola()">link</a>
</body>
</html>
```

En este ejemplo, hemos definido una función, llamada hola. Una función se declara así:

```
function (parámetros)
/* los parámetros separados por comas, sin escribir su tipo (booleano, numérico, ...) */
{
        instrucciones
}
```

En este caso, lo que hace esta función es llamar a la función de JavaScript alert, que muestra una ventana con el texto que se le pasa como parámetro.

Y, ¿ cuando se llama a esta función ?. Pues cuando ocurre el evento OnMouseOver del link, es decir, cuando el ratón pasa por encima del link. Eso viene marcado por la línea link

Como vemos, la filosofía de JavaScript es que en el script definimos funciones (en el head del documento) que serán llamadas cuando ocurran ciertos eventos.

3. Variables, operadores y estructuras de control

JavaScript permite utilizar variables, de tipos numéricos, cadenas de caracteres o booleanos. No obstante, cuando declaramos una variable no se le asigna un tipo, sino que es el propio intérprete el que, según el valor que se le asigne, le da un tipo u otro. La declaración es opcional, de manera que se puede hacer de dos maneras:

- Simplemente asignándole directamente un valor, por ejemplo, `x = 42`
- Utilizando la palabre clave `var`, por ejemplo, `var x = 42`

Por lo demás, la sintaxis del lenguaje es prácticamente igual que la de Java (es decir, igual que la de C). Así, tanto los operadores como las estructuras de control son exactamente iguales a C. Recordemos las estructuras de control:

Estructuras condicionales

- La estructura **if** ... **else** ... es así:

```
if (condición)
{
        bloque de instrucciones 1
}
else
{
        bloque de instrucciones 1
}
```

- La estructura **switch** es:

```
switch (expresión)
{
```

```
        case caso1 :
                instrucciones;
                break;
        case caso2:
                instrucciones;
                break;
        ...
        default : instrucciones;
}
```

Estructuras iterativas

- La estructura **while** es así:

```
while (condición)
{
        instrucciones
}
```

- La estructura **do** ... **while** es así:

```
do {
        instrucciones
} while (condición)
```

- La estructura **for** es así:

```
for (expresión inicial; condición de cumplimiento; expresión de incremento)
{
        intrucciones
}
```

4. Estructura de objetos

JavaScript es un lenguaje de objetos. Un objeto es un ente abstracto que agrupa por un lado a un conjunto de propiedades que definen al propio objeto y por otro, una serie de métodos que interactúan sobre él (funciones o procedimientos). Un ejemplo de objeto en un entorno de gestión de una empresa podría ser un empleado. Este empleado tiene una serie de propiedades, de datos que lo identifican: nombre y apellidos, sexo, DNI, número de seguridad social, fecha de nacimiento, estado civil, categoría, ... Por otro lado, hay una serie de acciones que se pueden hacer con un empleado, como contratar un nuevo empleado, despedir a un empleado ya existente, pagarle la nómina, subirle de categoría, ...

En JavaScript, se identifican una serie de objetos sobre los que podremos interactuar, como las ventanas, las páginas, las imágenes, los formularios, ... Por ejemplo, El objeto ventana tendrá, entre otras, una propiedad que sea el ancho de la ventana. Un método (funciones en el caso de JavaScriupt) asociado a este objeto será la función que abre una nueva ventana.

En los ejemplos anteriores, *document* es un objeto que representa a un documento HTML y *write* es un método que escribe la cadena que se pasa como parámetro. En el

otro ejemplo, *window* es un objeto que representa la ventana del navegador, una de cuyas propiedades es *status*, que al modificarse, lo que ocurre es que cambia la barra de status de la ventana. Tanto los métodos como las propiedades siempre se representan de la siguiente manera:

```
nombre_objeto.nombre_método(argumentos)

nombre_objeto.nombre_propiedad
```

donde, argumentos son los parámetros que se pasan a la función.

Hay que hacer notar que En JavaScript no hay procedimientos, sino que sólo hay funciones. De todos modos, hay funciones que retornan valores utilizando un return, pero también podemos optar por definir una función que no devuelva nada, lo que es similar a un procedimiento.

Veamos como se referencian las propiedades: tenemos un objeto llamado profesor con tres propiedades, que son el nombre, el año de nacimiento y el DNI:

```
profesor.nombre = “Toni Navarrete”
profesor.anyo_nacimiento = 1973
profesor.DNI = “12345678A”
```

Para crear nuestros propios objetos, lo hacemos siguiendo estos dos pasos:

1. Definir el objeto con una función constructor:

```
function profesor(nombre,anyo_nacimiento,DNI) {
        this.nombre = nombre;
        this.anyo_nacimiento = anyo_nacimiento;
        this.DNI = DNI;
}
```

this hace siempre referencia al objeto actual.

2. Crear una instancia del objeto usando *new*:

```
profe = new profesor(“Toni Navarrete”,1973,”12345678A”)
```

Y veamos cómo asignar métodos a un objeto. Para ello le asignamos una función que calcule los años del profesor. Para hacerla más simple, sólo restará el año actual menos el año de nacimiento.

1. Primero definimos la función:

```
function calcula_edad( ) {
        var edad = 2000 - this.anyo_nacimiento;
```

```
                /* las variables se declaran explícitamente,
                si bien el tipo se les asigna implícitamente */
        document.write("Su edad es: <b>" + edad + "</b>)
}
```

Realmente, la función se podría escribir en una línea, pero así vemos cómo declarar una variable local.

2. Asignamos la función al objeto, al crear el constructor:

```
function profesor(nombre,anyo_nacimiento,DNI) {
                /* nótese que no incluimos el nombre de la función */
        this.nombre = nombre;
        this.anyo_nacimiento = anyo_nacimiento;
        this.DNI = DNI;
        this.calcula_edad = calcula_edad( );
}
```

Y después de declarar las instancias utilizando new, llamaremos a la función desde el código HTML así:

```
profe.calcula_edad( )
```

5. Los objetos propios de JavaScript

Además de los objetos que podamos crear nosotros, JavaScript tiene una larga lista de objetos predefinidos y que nos serán de gran utilidad a la hora de añadir funcionalidades a nuestras páginas.

Estos objetos están agrupados formando una jerarquía que estudiamos a continuación:

En todas las páginas HTML encotraremos los objetos siguientes:

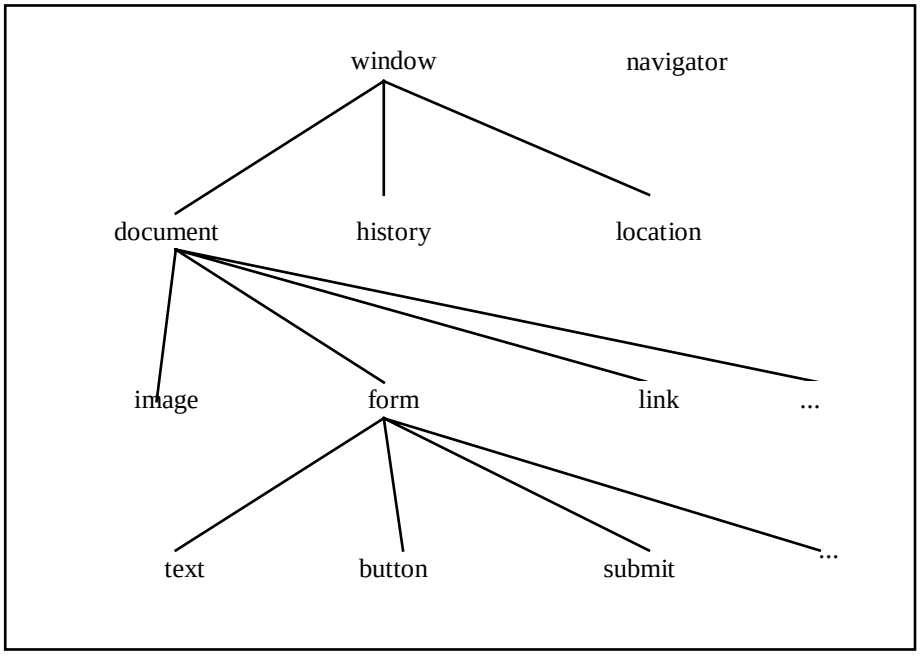
navigator: contiene propiedades como el nombre y la versión, los tipos MIME y plugins del navegador que utilizamos.

window: contiene las propiedades referentes a toda la ventana, o a un frame. Contiene a los siguientes tres objetos:

document: agrupa las propiedades del documento actual, tales como el título, formularios, imágenes, colores de texto o fondo, ...

history: contiene las URLs que el cliente ha visitado anteriormente

location: propiedades de la URL de la página



***5.1. El objeto* document**

Veamos un ejemplo:

```
<HTML>
<HEAD>
<TITLE>Ejemplo</TITLE>
</HEAD>
<BODY BGCOLOR="#000000" FGCOLOR="#FFFFFF">
<FORM NAME="mi_formulario" ACTION="procesa_formulario( )" METHOD="get">
Escribe tu texto: <INPUT TYPE="text" NAME="texto1" VALUE="Hola" SIZE=25><br>
</FORM>
<CENTER>
<IMG SRC="mi_imagen1.jpg"><IMG SRC="mi_imagen2.jpg">
</CENTER>
</BODY>
</HTML>
```

Veamos algunas propiedades:

```
document.fgColor = #FFFFFF
document.bgColor = #000000
document.title = Ejemplo
location.href = www.iua.upf.es/~tnavarrete/ejemplo.html
location.host = www.iua.upf.es
history.length = 5
document.mi_formulario.action = procesa_formulario( )
document.mi_formulario.method = get
```

```
document.mi_formulario.texto1.name = texto1
document.mi_formulario.texto1.value = Hola
```

En el siguiente ejemplo se muestra una página que al pulsar sobre los links, se cambia el color del fondo, cambiando la propiedad document.bgColor:

```
<HTML>
<HEAD>
<TITLE>Untitled-4</TITLE>
<script language="JavaScript">


function colorfondo(color)
{
      document.bgColor=color
}

</script>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<a href="#" onClick=colorfondo("#000000");>Pulsa para cambiar a negro el color de fondo</a><p>
<a href="#" onClick=colorfondo("#FFFFFF");>Pulsa para cambiar a blanco el color de fondo</a><p>
<a href="#" onClick=colorfondo("#FF0000");>Pulsa para cambiar a rojo el color de fondo</a><p>
<a href="#" onClick=colorfondo("#00FF00");>Pulsa para cambiar a verde el color de fondo</a><p>
<a href="#" onClick=colorfondo("#0000FF");>Pulsa para cambiar a azul el color de fondo</a><p>
<a href="#" onClick=colorfondo("#FFFF00");>Pulsa para cambiar a amarillo el color de fondo</a><p>

</BODY>
</HTML>
```

Volviendo al ejemplo anterior, todos los formularios de un documento están almacenados en un array, ordenado por un índice que empieza en 0 y se incrementa en uno, cada vez que creamos uno nuevo. Dicho array se llama *forms*. Por ejemplo:

```
document.forms[0].method = get
```

Lo mismo ocurre con la imágenes (array *images*), links (array *links*) anchos (*anchors*), applets de Java (*applets*). De igual modo, también hay un array, llamado *elements*, que almacena todos los elementos de un formulario.

```
document.images[0] hace referencia a la imagen mi_imagen1.jpg
document.images[1] hace referencia a la imagen mi_imagen2.jpg
```

Veamos un ejemplo donde se utiliza el array de imágenes para cambiar una imagen por otra al pasar el ratón por encima:

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
function carga(nom)
{
        nombo = nom + "2.gif";
        document.images[nom].src = nombo;
}

function carga(nom)
{
        nombo = nom + ".gif";
        document.images[nom].src = nombo;
}
</SCRIPT>
</HEAD>
<BODY>
<a href="pagina1.html" onmouseover="carga('uno')"
onmouseout="descarga('uno')";> <img src="uno.gif" name="uno" border=0></a>
<p>
<a href="pagina2.html" onmouseover="carga('dos')"
 onmouseout="descarga('dos')";> <img src="dos.gif" name="dos" border=0></a>
</BODY>
</HTML>
```

Aquí tenemos dos imágenes uno.gif y dos.gif. Al pasar por encima de uno.gif, se carga uno2.gif, y cuando retiramos el ratón se vuelve a cargar uno.gif. Lo mismo sucede con la otra imagen.

5.2. El objeto location

La propiedad más importantes es:

href: indica la URL actual completa.

De hecho, si no se indica propiedad alguna, se hace referencia a href. Por ejemplo:

```
window.location = “http://www.upf.es” , es similar a:
window.location.href = “http://www.upf.es”
```

Otras propiedades indican, por ejemplo, el nombre del servidor (propiedad *host*) o del puerto (propiedad *port*)

Tiene dos métodos

replace: carga el URL actual sobre la última ventana de la historia
reload: hace que se vuelva a cargar la ventana actual

5.3. El objeto history

Contiene una lista con las URLs que el navegador ha visitado. Las propiedades *history.current*, *history.next* y *history.previous* hacen referencia a las entradas actual, siguiente y anterior, respectivamente.

Para las demás, se puede hacer utilizando un índice; por ejemplo: history(3).

El método *go* permite cargar el contenido de una entrada concreta: history.go(3). en concreto history.go(0) carga de nuevo la página actual.

5.4. El objeto navigator

El objeto *navigator* contiene información sobre la versión del navegador que se está utilizando. Por ejemplo, la propiedad appName especifica el nombre del navegador, mientras que la propiedad appVersion nos da información sobre la versión de dicho navegador.

En cuando a sus métodos, pueden ser de interés javaEnabled que especifica si Java está habilitado en el navegador y preference que permite ver y modificar algunas preferencias del usuario (este último método apareció con la versión 1.2 de JavaScript).

Un caso típico del uso del objeto *navigator* es cuando necesitamos detectar si el browser del usuario es uno u otro (debido a problemas de compatibilidad). El siguiente ejemplo detecta si el navegador es un Netscape 4 o posterior:

```
if ((navigator.appName=='Netscape') && (navigator.appVersion.indexOf('Win')>-1) &&
(parseInt(navigator.appVersion)<=4) )
{
        alert("Tu navegador es Netscape de Windows, versión 4 o posterior")
        ...
}
```

También es importante hacer notar que el objeto *navigator* tiene dos objetos hijo, que son *Plugin* y *MimeType*, para interactuar con los plugins y diferentes tipos MIME.

5.5. Objeto window

Como aspecto importante de notación, todos los métodos y propiedades asociados al objeto *window* no es necesario que antepongan dicho nombre al nombre del método o propiedad. Por ejemplo, basta escribir document.write("hola") en vez de window.document.write("hola"), o open("hola.html") en vez de window.open("hola.html").

El objeto *window* tiene varios métodos dedicados a crear cajas de diálogo y nuevas ventanas. Veamos los más empleados:

El método *alert* abre una ventana de aviso (con sólo la opción de "Aceptar"), mientras que el método *confirm*, abre una ventana de diálogo con confirmación (con dos opciones).

Por otra parte, el método *open* abre una ventana nueva. Los parámetros son:

```
window.open("URL","nombre de la nueva ventana","parámetros auxiliares")
```

los parámetros auxiliares son:

- *toolbar = yes/no* (o tambíen 1/0): la barra de los botones Back, Forward, Stop, ...
- *location = yes/no* (o tambíen 1/0): la barra donde se escribe la URL
- *directories = yes/no* (o tambíen 1/0): la barra con What's New, Search, ...
- *status = yes/no* (o tambíen 1/0): la barra inferior de la ventana, donde se escribe el % que se ha cargado del fichero, su peso, ...
- *menubar = yes/no* (o tambíen 1/0): la barra de menús
- *scrollbars = yes/no* (o tambíen 1/0): las barras de scroll
- *resizable = yes/no* (o tambíen 1/0): indica si la ventana se puede cambiar de tamaño
- *width* = pixels: ancho de la ventana en pixels
- *height* = pixels: alto de la ventana en pixels

En los siete primeros, los del tipo *yes/no*, el valor por defecto es *yes*. Pero es importante saber que si se cambia uno de ellos, todos los demás pasan a ser automáticamente falsos, a no ser que se especifique lo contrario, poniéndolos (uno a uno) a *yes*.

También es importante aclarar que los parámetros auxiliares van separados por comas, y no debe haber ningún espacio en blanco entre ellos y las comas.

Veamos algún ejemplo:

`open("http://www.upf.es","UPF")` crea una ventana llamada UPF en la que se cargará http://www.upf.es

`open("http://www.upf.es","UPF","toolbar=yes,location=yes,directories=no")` crea una ventana similar a la anterior, pero donde no aparecerá la barra de directorios pero sí las de *location* y *toolbar*.

`open("http://www.upf.es","UPF","width=100,height=100")` crea una ventana como la primera, pero de tamaño 100x100.

En el siguiente ejercicio se crea una ventana flotante con contenidos dinámicos:

```
<HTML>
<HEAD>
```

```
<SCRIPT LANGUAGE="JavaScript">
function ImgOpen(nom,titol) {
    msg=window.open("","_blank","toolbar=no,directories=no,menubar=no,status=yes,wi
    dth=200,height=200");

    msg.document.open();

    msg.document.write("<HTML><HEAD><TITLE>"+titol+"</TITLE></HEAD>");
    msg.document.write("<BODY BGCOLOR='#FFFFFF'>");
    msg.document.write("<CENTER><img src=" + nom +"></CENTER>");
    msg.document.write("</BODY></HTML>");

    msg.document.close();
}
</SCRIPT>
</HEAD>
<BODY>
<a href="javascript:ImgOpen('rosa.gif','Rosa')>Pulsa para ver una rosa</a>
<a href="javascript:ImgOpen('clavel.gif','Clavel')>Pulsa para ver un clavel</a>
</BODY>
</HTML>
```

En este ejemplo, al clickar sobre rosa, se llama a la función ImgOpen, la cual crea una nueva ventana de 200x200 pixels. En esta nueva ventana, vamos escribiendo dinámicamente, utilizando document.write, la página HTML. Usamos como parámetro la imagen que debe mostrar én la nueva ventana y el título que tendrá la ventana.

Cada frame (marco) de una página resulta ser un objeto del tipo *window*. Todos los frames, al igual que pasaba con los formularios de un documento, están almacenados en un array denominado *frames*. Conforme se crean las etiquetas *frameset* en los documentos HTML, así se llena dicho array. Nótese que este array puede ser recursivo, ya que dentro de un *frame* se pueden definir nuevos.

Vemos un ejemplo:

```
<FRAMESET COLS=20%,*>
        <FRAMESET ROWS= 25%,*>
                <FRAME SRC="derecha_arriba.html">
                <FRAME SRC="derecha_abajo.html">
        </FRAMESET>
        <FRAMESET ROWS= 25%,*>
                <FRAME SRC="izquierda_arriba.html">
                <FRAME SRC="izquierda_abajo.html">
        </FRAMESET>
</FRAMESET>
```

El array *frames* quedará de la siguiente manera:

```
frames[0] = derecha_arriba.html
frames[1] = derecha_abajo.html
frames[2] = izquierda_arriba.html
frames[3] = izquierda_abajo.html
```

Si dentro de izquierda_abajo volvemos a dividir en dos frames, uno.html y dos.html accederemos así, respectivamente:

frames[3].frames[0] y frames[3].frames[1]

Hay dos propiedades especiales que son parent y top:

parent indica la ventana (o frame) que contiene a la actual. Por ejemplo, si estamos en el fichero uno.html, parent.frames[1] hace referencia al fichero dos.html.

top indica la ventana primera de todas (la mayor). Por ejemplo, si seguimos estando en el fichero uno.html, top.frames[1] hace referencia al fichero derecha_abajo.html.

A continuación se presenta un ejemplo donde se explica cómo se pueden actualizar dos frames a la vez:

Veamos primero cómo definimos los frames: tenemos un frame abajo que ocupa 100 pixels, y la parte de arriba, ocupando el resto, la tenemos dividida en dos frames al 50%. Ésta es el fragmento de la página donde se hace la división en tres frames es:

```
<FRAMESET ROWS="*,100" >
        <FRAMESET COLS="50%,*" >
                <FRAME SRC="izda.html" name="izda" NORESIZE>
                <FRAME SRC="dcha.html"  name="dcha" NORESIZE>
        </FRAMESET>
         <FRAME SRC="abajo.html"  name="abajo" NORESIZE>
</FRAMESET>
```

El array de frames ha quedado así:

frames[0] hace referencia a izda. También podemos escribir frames[‘izda’]
frames[1] hace referencia a dcha. También podemos escribir frames[‘dcha’]
frames[2] hace referencia a abajo. También podemos escribir frames[‘abajo’]

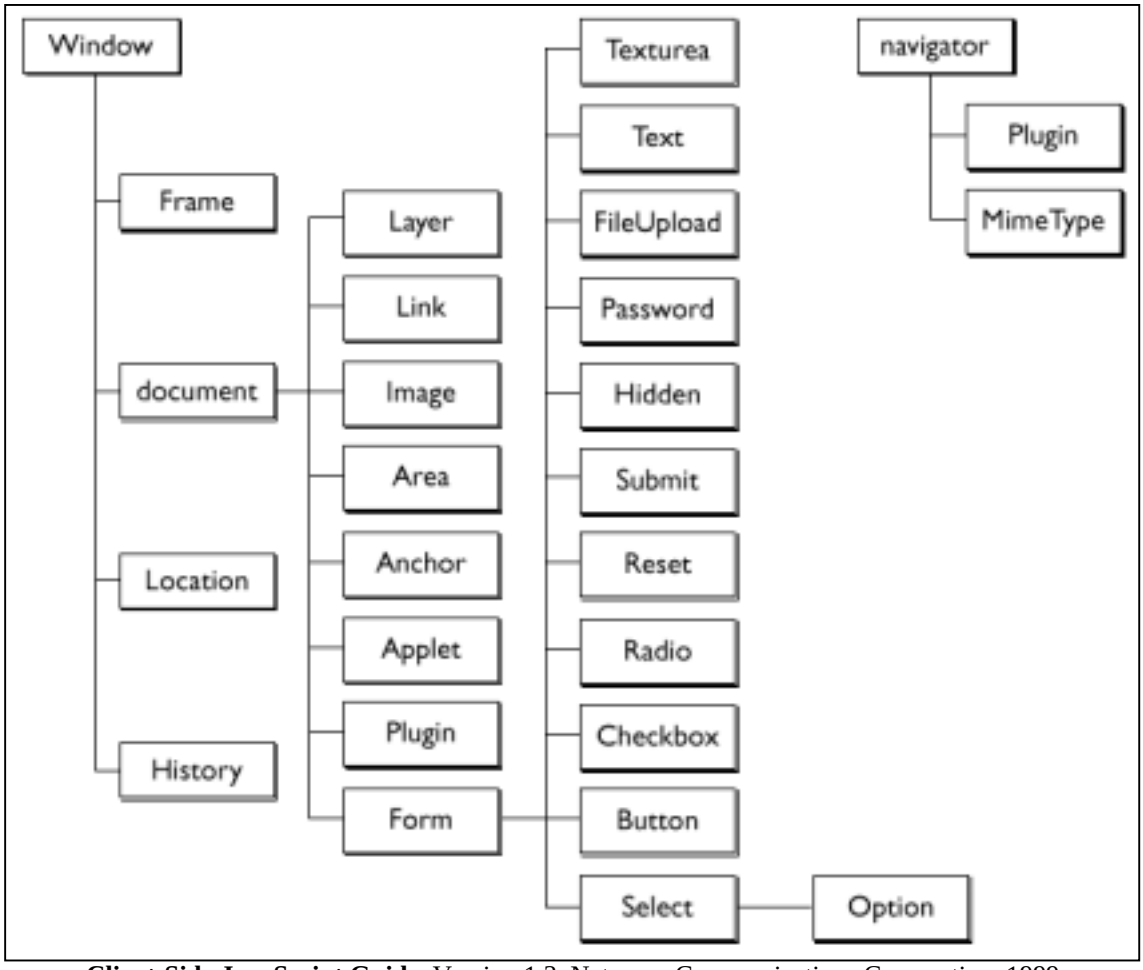
Ahora queremos que en el frame de abajo, al pulsar sobre el link, cambien los dos frames de arriba. Para ello tenemos que crear una función que cambie la propiedad location.href de ambos:

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
function cambia_2frames( )
{
        frames[0].location.href = 'izda2.html';
        frames[1].location.href = 'dcha2.html';
}
</SCRIPT>
```

```
</HEAD>
<BODY>
<a href="javascript:cambia_2frames( )">Pulsa y cambiaras los dos frames de arriba</a>
</BODY>
</HTML>
```

5.6. El modelo de objetos completo

En el siguiente diagrama aparece el modelo de objetos de JavaScript versión 1.3, al completo:



Client-Side JavaScript Guide, Version 1.3. Netscape Communications Corporation, 1999

Apéndice. Referencias interesantes

El site central de JavaScript es el de Netscape, en sus páginas de desarrolladores (http://developer.netscape.com). Allí se encuentra el JavaScript Developer Central, donde se pueden encontrar documentación, tutoriales, herramientas, libros, artículos, FAQ, newsgroup, bugs conocidos, enlaces interesantes, ... :

http://developer.netscape.com/tech/javascript/index.html

Dynamic HTML Developer Central, es la web de desarrolladores de Netscape sobre HTML dinámico:

http://developer.netscape.com/tech/dynhtml/index.html

Voodoo's Introduction to Javascript, por Stefan Koch

http://rummelplatz.uni-mannheim.de/~skoch/js/tutorial.htm, que se encuentra traducido al castellano por J. Mauricio Cuenca, aunque sólo en sus primeros capítulos en: http://200.25.9.3/enlaces/javascript/index.html