El lenguaje JavaScript

Asignatura: Fonaments Cartografia i SIG
Curso: 2006/07
Profesor: Toni Navarrete

1. Introducción

JavaScript es 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 (Mozilla Firefox, Microsoft Internet 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. 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".

JavaScript fue declarado como estándar del European Computer Manufacturers' Association (ECMA) en 1997, y poco después también fue estandarizado por ISO. Sin embargo, la estructura de objetos que implementaban los diferentes navegadores (Netscape y Explorer en aquellos momentos) no se ajustaba al estándar, lo que provocaba numerosos problemas de compatibilidad. Para solventarlos el W3C publicó un nuevo modelo de objetos, DOM (Document Object Model), que incorporan la mayoría de navegadores actuales como Explorer o Firefox.

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 básico de una página HTML que contiene un script de Javascript:

```
<HTML>
<BODY>
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 diferentes navegadores). Ésta es una lista más extensa de eventos:

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,	User releases a mouse button	`onMouseUp`

	buttons, links		
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

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 JavaScript) 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.

Veamos cómo 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 = 2007 - 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( )
```

Antes hemos dicho que JavaScript es un lenguaje de objetos ya que permite trabajar con una serie de objetos. Sin embargo no es exactamente un lenguaje orientado a objetos como lo son Java o C++, ya que JavaScript no soporta herencia. Suele decirse que JavaScript es un lenguaje basado en prototipos, ya que las nuevas clases se generan clonando las clases base (prototipos) y extendiendo su funcionalidad. La palabra reservada *prototype* hace referencia al prototipo (clase) de un objeto. Por ejemplo, podemos añadir nuevas propiedades y métodos a una clase de la siguiente manera:

```
profe.prototype.nuevaPropiedad=123;
profe.prototype.nuevoMetodo=f;
```

donde f es una función ya definida. Nótese que se pone f (sin paréntesis), ya que lo que se iguala es la referencia de la función y no el valor que devuelve.

Este mecanismo es el que se utiliza para definir una cierta herencia. Por ejemplo, podríamos definir una nueva clase *Profesor2* que hereda de *Profesor* y añade el método *menor30*, de la siguiente manera:

```
function menor30() {
        if (this.anyo_nacimiento<1977)
                return 1;
        else
                return 0;

}

function Profesor2() {
}

Profesor2.prototype = new Profesor();
Profesor2.prototype.menor30 = menor30;
```

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 encontraremos 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

Como ejemplo, el script *colorfondo* cambia la propiedad *bgColor* del objeto *document*, lo cual hace que se cambie el color de fondo de la página, en función del parámetro que se le envía:

```
<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>
```

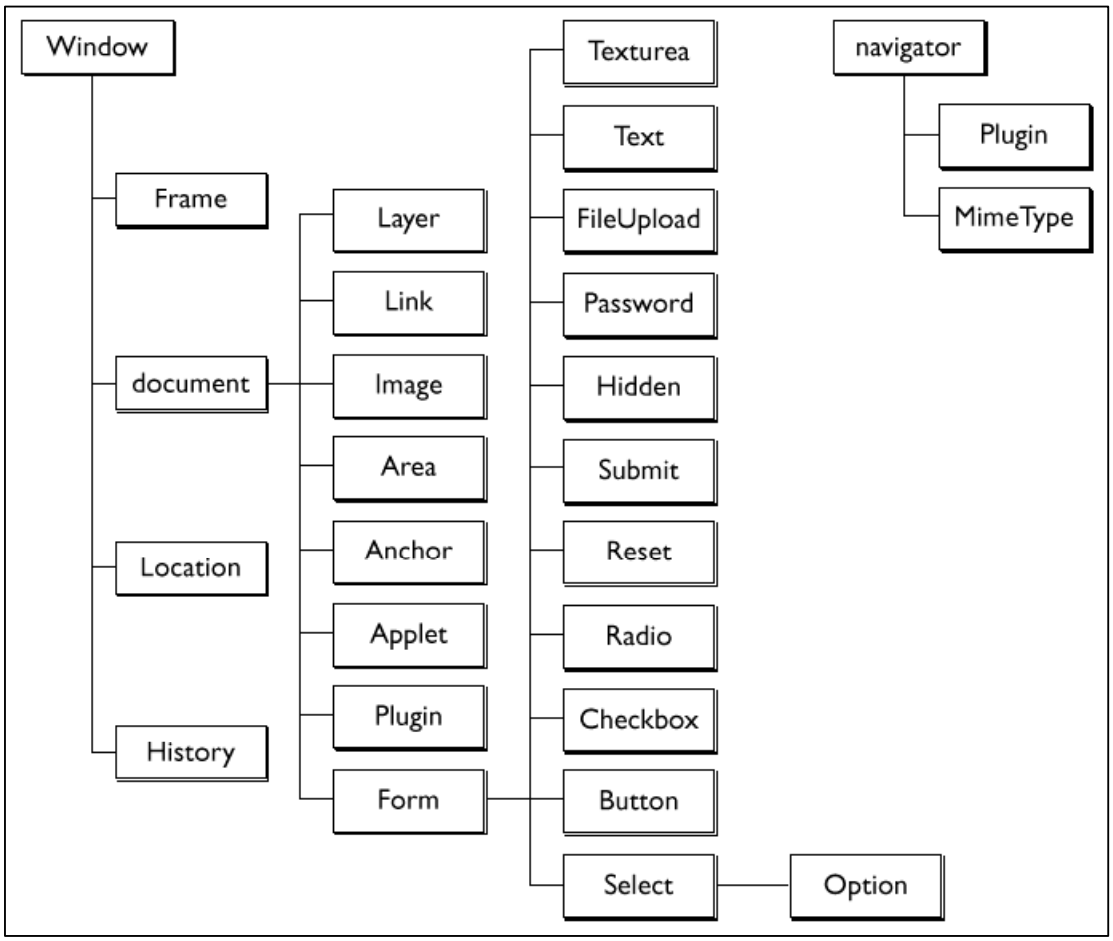
El siguiente ejemplo incluye la función *carga*, que cambia la imagen (‘uno.gif’ o ‘dos.gif’) al pasar por encima el ratón, y en su lugar carga las imágenes ‘uno2.gif’ o ‘dos2.gif’, respectivamente. La función *descarga* hace el proceso inverso, cuando el ratón sale de encima de la imagen. Nótese que la función carga se llama al producirse el evento *onmouseover*, mientras que descarga con el evento *onmouseout*.

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

function carga(nom)
{
        nombo = nom + "2.gif";
        document.images[nom].src = nombo;
}

function descarga(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>
```

En el siguiente diagrama aparece el modelo de objetos de JavaScript ECMA:



Para una referencia del modelo DOM del W3C: http://www.w3.org/DOM/