El lenguaje JavaScript

JavaScript es un lenguaje de scripts compacto basado en objetos (y no orientado a objetos). Originariamente era denominado *LiveScript*, y fue desarrollado por Netscape para su navegador *Netscape Navigator 2.0*. Fue éste el primer cliente en incorporarlo. Se ejecuta sobre 16 plataformas diferentes, incluyendo los entornos de Microsoft e incluso el MS Explorer lo incorpora en su versión 3.0 .

JavaScript permite la realización de aplicaciones de propósito general a través de la WWW y aunque no está diseñado para el dessarrollo de grandes aplicaciones es suficiente para la implementación de aplicaciones WWW completas o interfaces WWW hacia otras más complejas..

Por ejemplo, una aplicación escrita en JavaScript puede ser incrustada en un documento HTML proporcionando un mecanismo para la detección y tratamiento de eventos, como clicks del ratón o validación de entradas realizadas en *forms.*

Sin existir comunicación a través de la red una página HTML con JavaScript incrustado puede interpretar, y alertar al usuario con una ventana de diálogo, de que las entradas de los formularios no son válidas. O bien realizar algun tipo de acción como ejecutar un fichero de sonido, un applet de Java, etc.

JavaScript y Java

Las diferencias entre Java y JavaScript son notables pero tambien sus similitudes.

En primer lugar Java es un lenguaje de programación mientras que JavaScript es un lenguaje de scripts (como su nombre indica). Éste último es más sencillo de entender y usar que Java si no se tienen conocimientos previos de metodología de programación orientada a objetos. La mayor sencillez de JavaScript hacen que sea interesante aprender éste último lenguaje como paso previo a adentrarse en el mundo de Java.

JavaScript es mucho más modesto pero precisamente por ello es más sencillo. Se basa en un modelo de instanciación de objetos muy simple para el que no es necesario tener conocimiento de conceptos tales como herencia y jerarquías.

Soporta un sistema en tiempo de ejecución basado en un pequeño número de tipos de datos (numérico, Boolean, y string) en el que ni siquiera es necesario declarar el tipo de variables. Sin embargo Java exige una gran rigidez en el tipo de datos utilizados y dispone de una amplia variedad de tipos básicos predefinidos, operadores y estructuras de control.

En Java uno de los principales bloques de programación son las clases a las que se asocian funciones específicas. Para utilizarlas es necesario instanciarlas en objetos. Los requerimientos de Java para declarar dichas clases, diseñar sus funciones, y encapsular tipos hacen que la programación en este lenguaje sea mucho más compleja que la realizada con JavaScript.

Otra diferencia importante es que Java es un lenguaje lo bastante potente como para desarrollar aplicaciones en cualquier ámbito. No es un lenguaje para programar en Internet, sino que se trata de un lenguaje de propósito general, con el cual se puede escribir desde un applet para una página Web (ésto es es una pequeña aplicación escrita con un determinado formato que se ejecuta en un trozo de un documento HTML) hasta una aplicación que no tenga ninguna clase de conexión a Internet.

Los requerimientos tambien son diferentes; Para programar en JavaScript sólo es necesario un editor de texto mientras que para programar en Java se necesita un compilador específico.

La complejidad de Java es semejante a la de un programa en C++ mientras que la de JavaScript es cercana a la de uno en dBase, Clipper o sh.

Por otra parte, la sintaxis de ambos lenguajes es muy similar sobre todo en lo que a estructuras de control de flujo se refiere.

Existen además mecanismos de comunicación entre Java y JavaScript.

En definitiva, la principal ventaja de JavaScript es su simplicidad y su menor demanda de requisitos.

Relación entre JavaScript y Java:

JavaScript	Java
Interpretado (no compilado) en cliente.	Compilado en servidor antes de la ejecución el el cliente.
Basado en objetos. Usan objectos, pero no clases ni herencia.	Programación orientado a objetos. Los applets constan de clases objeto con herencia.
Codigo integrado en el código HTML.	Applets diferenciados del código HTML (accesibles desde las páginas HTML).
No es necesario declarar el tipo de las variables.	Necesario declarar los tipos.
Enlazado dinámico. Los objetos referenciados deben existir en tiempo de ejecución (lenguaje interpretado).	Enlazados estáticos. Los objetos referenciados deben existir en tiempo de compilación (lenguaje compilado).

JavaScript y CGI

CGI (the *Common Gateway Interface*) es una interfaz entre programas de aplicación y servicios de información. Es decir, son un conjunto de reglas a cumplir tanto por parte del servidor como por parte del programa, pero se deja libertad al programador a la hora de escoger el lenguaje que considere mas adecuado para programar la aplicación. Un programa en CGI puede ser escrito en cualquier lenguaje como: C/C++ , Fortran, PERL ,TCL, etc.

En JavaScript no existen restricciones a cumplir en el Servidor hasta el punto que ni siquiera es necesario que éste exista.

Por otra parte y al contrario que CGI, JavaScript únicamente depende del cliente y no del sistema operativo, sólo necesita un browser capaz de interpretarlo. Cualquier persona puede desarrollar aplicaciones escritas en JavaScript del mismo modo que realiza páginas HTML. Esto no ocurre con aplicaciones CGI que necesitan la existencia de un servidor WWW para ser ejecutadas.

Con JavaScript todo el código es trasladado al cliente y no se necesita la comunicación a través de la red cada vez que se produce un evento, como se requería en CGI.

Por otro lado, JavaScript no es un lenguaje válido para desarrollar aplicaciones concurrentes y/o de acceso compartido.

Comparación entre JavaScript y CGI:

JavaScript	**CGI**
Es un lenguaje de programación	Es una interfaz. Da libertad de elección del lenguaje
No requiere un servidor de WWW	Exige la presencia de un servidor WWW
No requiere una red, funciona en local.	Requiere comunicación en red. No funciona en local
La aplicación reside en el cliente	La aplicación reside en el servidor
Requiere un cliente especial	Sirve cualquier cliente, por simple que sea
Pensado para aplicaciones monousuario	Permite desarrollo de aplicaciones distribuidas, acceso concurrente y/o compartido
Tiempo de desarrollo muy breve	Tiempo de desarrollo medio

JavaScript y HTML

Los programas en JavaSecript aparecen incrustados en los propios documentos HTML como si de HTML se tratara. Pueden integrarse de dos formas:

- Como programas propiamente dichos, combinando funciones y sentencias, con el mismo aspecto que tendría el código de cualquier otro lenguaje.
- Introduciendo manejadores de eventos JavaScript en etiquetas HTML.

Presentaremos los dos mecanismos:

La etiqueta script

La manera más convencional en que aparece JavaScript en un documento es en forma de programa. Podemos empezar por mostrar unos breves scripts y ver como son implementados dentro de documentos HTML. Empezaremos con un pequeño programa que muestra un texto en un documento HTML.

```
<html>
<head>
¡Mi primer JavaScript!
</head>
<body>
<br>
Este es un documento HTML normal
<br>
<script language="JavaScript">
document.write("Esto es JavaScript!")
</script>
<br>
En HTML otra vez.
</body>
</html>
```

Este primer programa se limita a escribir en pantalla un determinado texto para lo que se emplea el código *document.write*. En este código, *document* es un objeto creado por el sistema que hace referancia al propio documento y *write* es uno de los métodos que proporciona para interactuar con él. El resultado de cargar este documento en un browser que interprete JavaScript será la aparición de los dos textos, el escrito en JavaScript y el escrito en HTML, sin que el usuario sea consciente del proceso.

El resultado seria:

Este es un document HTML normal.

Esto es JavaScript!

En HTML otra vez

Este script no es muy útil pero sirve para mostrar el uso de la etiqueta <SCRIPT>. Puedes usar estas etiquetas en cualquier lugar del documento, tanto en la cabecera como en el cuerpo, aunque si se declaran funciones es más aconsejable hacerlo en la cabecera.
La etiqueta <SCRIPT>es una extensión de HTML en la que se encierra el texto que compone el código del programa JavaScript correspondiente de la manera siguiente:
<SCRIPT>

Sentencias JavaScript...

</SCRIPT>
De esta manera el navegador que "entienda" JavaScript reconocerá el texto encerrado entre estas etiquetas como código JavaScript y no lo mostrará en la pantalla del cliente. Una cuestión importante a considerar es el mantenimiento de la compatibilidad con navegadores anteriores. Cualquier browser ignora las etiquetas desconocidas, por lo tanto, aquellos que no soporten JavaScript ignorarán el comienzo y el final del código del programa (encerrado entre las etiquetas <SCRIPT> y </SCRIPT>).Para que el resto del código tambien sea ignorado y no aparezca en la pantalla del cliente, lo encerraremos entre los símbolos de comentario HTML, <!-- y -->.
Los navegadores que, por el contrario si lo soporten, interpretarán el código encerrado entre las etiquetas SCRIPT e ignorará el principio de la linea en el script que comienza con la doble slash (//) o bien el encerrado entre "/*" y "*/", que son los símbolos de comentarios en este lenguaje.
Un documento puede tener varias etiquetas SCRIPT, y cada una de ellas incluir sentencias JavaScript diferentes.
Si queremos dar la posibilidad de ejecutar un código alternativo a los browsers que no interpretan JavaScript, debemos utilizar las etiquetas <NOSCRIPT></NOSCRIPT>
Por ejemplo:

```
<SCRIPT>
<!-- Ocultación a browsers antiguos
document.write("Si ves esto, tu browser interpreta JavaScript")
// Fin de la ocultación -->
</SCRIPT>
</HEAD>
<BODY>
<NOSCRIPT>
Si ves esto, tu browser no incorpora la etiqueta
</NOSCRIPT>
</BODY>
</HTML>
```

Con vistas a un uso futuro, esta etiqueta admite un parámetro opcional LANGUAGE que indica el lenguaje de script que se ha incrustado en el documento así como la versión de JavaScript.

<SCRIPT LANGUAGE="*Versión de JavaScript*";>

Sentencies JavaScript...

</SCRIPT;>

Versión de JavaScript especifica la versión de JavaScript en la que está escrito el código, y puede ser:

- <SCRIPT LANGUAGE="JavaScript"> especifica JavaScript para Navigator 2.0.
- <SCRIPT LANGUAGE="JavaScript1.1"> especifica JavaScript para Navigator 3.0.

Las sentencias encerradas entre las etiquetas son ignoradas si el browser que las lee no tiene el nivel de JavaScript especificado en el atributo LANGUAGE; por ejemplo:

- Navigator 2.0 ejecuta el código escrito con la etiqueta <SCRIPT LANGUAGE="JavaScript">e ignora el código escrito en la etiqueta <SCRIPT LANGUAGE ="JavaScript1.1">.
- Navigator 3.0 ejecuta el código escrito entre las etiquetas <SCRIPT LANGUAGE="JavaScript"> o <SCRIPT LANGUAGE="JavaScript1.1">.

Si el atributo LANGUAGE es omitido, Navigator 2.0 asume LANGUAGE="JavaScript" y Navigator 3.0 asume LANGUAGE="JavaScript1.1"

Puede usar el atributo LANGUAGE para escribir scripts que contengan código para Navigator 3.0 (con nuevas funciones que no existían en versiones anteriores) y que no provoquen un error ejecutándose bajo Navigator 2.0.

Ejemplo. Este ejemplo muestra como usar dos versiones diferentes de JavaScript una para JavaScript 1.0 y otro para JavaScript 1.1. El documento cargado por defecto es para JavaScript 1.0. Si el usuario utiliza Navigator 3.0, utilizará la función location.replace("..") implementada únicamente en esta versión de JavaScript.

<SCRIPT LANGUAGE="JavaScript1.1">

location.replace("mipagina.html") //Sustituye la página actual por "mipagina.html"

</SCRIPT>

[Definición del resto de funciones compatibles...]

En el Netscape 3.0 se añade un nuevo parámetro opcional a la etiqueta <SCRIPT>, SRC. El objeto del mismo es el de actuar como un *include*, incluyendo en el documento un código JavaScript que pueda ser compartido por diversos documentos, es decir, posibilitar el uso de librerias. Se recomienda que éstas tengan extensión ".js". La sintaxis asociada será:

El atributo SRC debe especificar una URL, relativa o absoluta. Por ejemplo:

<SCRIPT SRC="libreria.js"></SCRIPT>

<SCRIPT SRC="http://home.netscape.com/functions/libreria.js">

Esta libreria no puede contener código HTML, únicamente definiciones de funciones JavaScript.

Funciones en JavaScript

Las funciones son unos de los bloques fundamentales en JavaScript. Una función es un procedimiento escrito en JavaScript, es decir, un conjunto de sentencias que realizan una determinada tarea.
Una función consta de las siguientes partes básicas:

- Un nombre de función.
- Los parámetros pasados a la función separados por comas y entre paréntesis.
- Marcar el inicio y el final de la función entre llaves ({})

Es importante entender la diferencia entre definir una función y llamarla.
Definir una función es simplemente especificar su nombre y definir que acciones realizará en el momento en que sea invocada. Para ello se emplea la palabra reservada *function.*

Function nombre_de _la_función([parámetro1, parámetro2,...])
{
.....
return <valor_retorno>
}

Llamando a esta función realizará las acciones especificadas con los parámetros indicados.

Nombre_de_la_función(a,b);

Las funciones pueden definirse en cualquier parte del documento pero es aconsejable declararlas en la cabecera; de esta forma se garantiza que todas las funciones se cargen antes se que el usuario tenga la oportunidad de realizar ninguna acción en la que llame a una de ellas.
El siguiente ejemplo define una función en la cabecera (HEAD) de un documento y la llama en el cuerpo (BODY).

```
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
<!-- Oculto el código a los browsers que no entiendan "JavaScript"
function cuadrado(numero) {

        return numero * numero

}
// Final de la ocultación-->
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT>
document.write("La función retorna ", cuadrado(5), ".")
</SCRIPT>
</BODY>
```

La función cuadrado tiene un argumento, llamado numero. La función consta de una sentencia

return numero * numero
que indica que ha de retornar el cuadrado del número pasado por parámetro.
En el cuerpo del documento hacemos una llamada a la función definida mediante la sentencia:
cuadrado(5)
La función ejecuta la sentencia *numero * numero* como *5 * 5* y retorna el valor 25, el script muestra el siguiente resultado en pantalla:
La función retorna 25.

Manejadores de eventos

La mayoria de las acciones de programa (al tratar con una aplicación de WWW) deben ser activadas por eventos. Los eventos son acciones que ocurren como resultado de alguna acción realizada por el usuario. Un click de ratón, la focalización de un campo en un formulario, modificar un campo de texto o mover el cursor son ejemplos de eventos (ver tabla).
La segunda forma de incrustación de JavaScript en documentos HTML consiste en la definición de manejadores de eventos en las etiquetas. La sintaxis general es:
<ETIQUETA
manejador_evento = "Código JavaScript">
Donde ETIQUETA es cualquier etiqueta HTML que pueda relacionarse con un evento y *manejador_evento*, el evento en sí.
Cada evento es es reconocido por ciertos objetos, etiquetas HTML, como son:

- **Focalización, desfocalización y edición**: *campos de texto*, *textareas* y *selections*.
- **Clicks**: *buttons*,*radio buttons*, *checkboxes*, *submit buttons*, *reset buttons*, y *enlaces*.
- **Selección**: *text fields*, *textareas*.
- **Señalización**: *enlaces*.

A partir de la versión 3.0 de Netscape, *onBlur* y *onFocus* se aplican tambien a ventanas y framesets.

El nombre de un manejador de eventos es el nombre del evento,precedido por "on.". Por ejemplo, el manejador de eventos para *focus* en *onFocus*.
La relación entre eventos y nombres de manejadores, así como su descripción , se puede observar en la tabla adjunta:

Evento	Manejador	Descripción
click	*onClick*	El usuario pulsa sobre un elemento de form o un enlace
señalización	*onMouseOver*	El usuario coloca el ratón sobre una determinada area
carga	*onLoad*	El usuario carga la página
descarga	*onUnload*	Es usuario sale de la página
focalización	*onFocus*	El usuario selecciona un

		elemento de un form como entrada
desfocalización	*onBlur*	El usuario quita la selección de un elemento de un formulario como entrada
edición	*onChange*	El usuario cambia el valor de un elemento text, textarea o select de un formulario
selección	*onSelect*	El usuario selecciona el campo de entrada de un formulario
Interrupción	*onAbort*	El usuario interrumpe la carga de una imagen
Error	*onError*	La carga de un documento o imagen produce un error.
Des-señalización	*onMouseOut*	El usuario saca el ratón de un área (imagemap) o enlace.
Inicialización	*onReset*	El usuario pulsa el botón de reset en un form.

Ejemplo. En el siguiente ejemplo definimos un campo de texto en el que entraremos un número, *x*. Al hacer click sobre el botón obtendremos el resultado (x^2) en el segundo campo de texto.

Al mismo tiempo, aparecerá un mensaje en la barra de status que indicará el número introducido así como el resultado obtenido tras aplicarle el cuadrado. Para realizar estas acciones utilizaremos la función definida más arriba como *cuadrado*.

Cada vez que el usuario haga un click sobre el botón *Cuadrado* (para determinarlo utilizaremos el manejador de eventos *onClick*) se llamará a la función del mismo nombre, pasándole como parámetro el valor introducido en el primer campo de texto (*x*)

Inmediatamente será mostrado en la zona de status un mensaje del tipo: *"Has entrado el número x y el resultado es x^2"* mediante la sentencia *window.status*,donde window es un objeto creado por el sistema que hace referencia a la ventana actual y *status* no es más que una de sus propiedades que especifica el mensaje a mostrar en la barra de estado.

El resultado es el siguiente:

Entra un número:

Resultado: 0

Y el código asociado:

```
<FORM>
Entra un número:

    <INPUT TYPE="text" NAME="entrada" SIZE=15>
    <INPUT TYPE="button" VALUE="Cuadrado"
    onClick="cuadrado(entrada.value)">
```

```
<BR>
Resultado:

        <INPUT TYPE="text" NAME="resultat" SIZE=15 VALUE = 0>

</FORM>
```

Las modificaciones realizadas en la función cuadrado son las siguientes:

```
function cuadrado(numero){

        document.form1.resultat.value = numero * numero;
        window.status='Has entrado el número ' + numero + ' y el resultado es '
        +document.form1.resultat.value;

}
```

Mediante los manejadores de eventos es posible dotar a los documentos HTML de una gran interactividad.