

JavaScript

Paso a paso

Edgar D'Andrea

Tabla de contenidos

1:	Introducción.....	15
2:	Conceptos básicos.....	31
3:	Programación orientada a objetos.....	59
4:	Objeto Array.....	71
5:	Objeto Date.....	81
6:	Objeto String.....	93
7:	Objeto Math.....	119
8:	Objeto window.....	133
9:	Cookies.....	159
10:	Objeto Image.....	171
11:	Gestión de eventos.....	175
12:	Gestión del teclado y del ratón.....	185
13:	Formularios.....	197
14:	DHTML y objeto document.....	225
15:	DOM.....	257
16:	Ajax y JavaScript.....	287
A:	Respuestas.....	305

Índice general

Capítulo 1:

Introducción.....	15
JavaScript, un lenguaje cliente.....	16
JavaScript no es Java	17
Lo que JavaScript no hace.....	17
La compatibilidad puesta en duda.....	19
Una decisión estratégica	20
¿Cómo codificar con JavaScript?	20
Complementos en los navegadores.....	21
Extensión para desarrollador Web.....	22
Herramientas de desarrollo de Internet Explorer.....	23
Transferencia de servidor local a remoto.....	24
Recomendaciones para el entorno de pruebas.....	25
Ejecución de una aplicación JavaScript.....	26
Conexión HTML/JavaScript mediante eventos.....	29
Resumen.....	30
Prácticas del capítulo 1.....	30

Capítulo 2:

Conceptos básicos.....	31
Estructura del código	33
Comentarios en el código.....	34
Nomenclatura	35
Nomenclatura de variables.....	35
Nomenclatura de funciones.....	36
Longitud de los nombres.....	36
Palabras reservadas.....	37
¿Está activado JavaScript?.....	37
Scripts de JavaScript.....	37
Modo indirecto.....	38

Uso de archivos externos.....	40
Bloque de instrucciones.....	41
Variables.....	41
Nomenclatura de las variables.....	42
Tipo de una variable.....	43
Sentencias condicionales.....	44
Sentencia if.....	44
Operadores de comparación de valores.....	45
Combinación de condiciones.....	45
Sentencia if/else.....	46
Sentencia switch.....	46
Operador ternario.....	47
Control de bucles de ejecución.....	47
Sentencia for.....	47
Incremento/decremento por unidades.....	49
Sentencia while.....	49
Sentencia do/while.....	50
Funciones.....	50
Parámetros de la función.....	51
Llamada de función.....	51
Cómo saber cuál fue la función llamante.....	52
Una función que se llama a sí misma.....	53
Ámbito de variables: locales y globales.....	54
Evaluación de una cadena de código JavaScript.....	55
JavaScript y los motores de búsqueda	56
Prácticas del capítulo 2.....	57

Capítulo 3:

Programación orientada a objetos.....	59
Objetos propios de JavaScript	60
Acceso a las propiedades de los objetos.....	61
Uso de métodos de un objeto	61
Sentencia for ...in para objetos.....	62
Sentencia with.....	63
Definición y creación de objetos en JavaScript.....	64

Cómo se define un método de un objeto.....	64
Puntero this para referenciar un objeto.....	65
Operador new para crear el objeto.....	65
Contención de objetos.....	66
Copia de objetos.....	67
Prácticas del capítulo 3.....	69

Capítulo 4:

Objeto Array.....	71
Necesidad del uso de arrays.....	71
Objeto Array().....	72
Las matrices son, por sobre todo, objetos.....	73
Matrices asociativas.....	74
Matrices multidimensionales.....	74
Métodos del objeto Array.....	75
Prácticas del capítulo 4.....	79

Capítulo 5:

Objeto Date.....	81
Objeto Date.....	82
Métodos de lectura.....	83
Métodos de escritura.....	85
Función setTimeout().....	87
Función setInterval().....	88
Ejemplo: Zona horaria internacional.....	88
Prácticas del capítulo 5.....	92

Capítulo 6:

Objeto String.....	93
Gestión de cadenas de texto.....	94
Uso de escapes.....	93
Concatenación de cadenas.....	94
Objeto String.....	94
Métodos más comunes.....	95
Métodos vinculados al formateo HTML.....	96

Acceso a los caracteres de una cadena.....	97
Código ASCII.....	98
La cuestión de la codificación de caracteres.....	99
Subcadenas.....	99
Búsqueda dentro de una cadena.....	100
Conversión a mayúsculas o minúsculas.....	100
Evaluación de una cadena.....	101
Escape de caracteres.....	101
Expresiones regulares.....	102
Definición de expresiones regulares.....	104
Codificación de la expresión regular.....	104
Método test() del objeto RegExp.....	106
Método exec() del objeto RegExp.....	107
Método match() del objeto String.....	108
Método split() del objeto String.....	109
Método replace() del objeto String.....	111
Prácticas del capítulo 6.....	116

Capítulo 7:

Objeto Math.....	119
Operaciones de conversión a número.....	120
No es un número (NaN).....	120
Operación inversa (de número a cadena).....	122
Objeto Math.....	122
División por cero.....	125
Constantes matemáticas.....	125
Formateo de números.....	126
Calculadora de ejemplo.....	127
Prácticas del capítulo 7.....	132

Capítulo 8:

Objeto window.....	133
Modelo BOM.....	133
Objeto window.....	134

Métodos del objeto window.....	135
Propiedades del objeto window.....	137
Métodos auxiliares para cuadros de diálogo.....	138
Objeto navigator.....	140
Objeto screen.....	143
Objeto location.....	144
Objeto history.....	147
Cambio de tamaño de la ventana.....	148
Cambio de posición de la ventana.....	149
Impresión de la pantalla.....	149
Ventanas emergentes (popup).....	150
Comunicación entre ventanas.....	152
Precaución al usar ventanas popups.....	155
Frames.....	155
Frames y la seguridad.....	156
Etiqueta iframe	157
Prácticas del capítulo 8.....	157
 Capítulo 9:	
Cookies.....	159
Cookies y seguridad.....	160
Limitaciones de las cookies	162
Interacción con otros lenguajes	163
Gestión de cookies.....	163
Escritura de cookies.....	163
Ejemplo.....	166
Eliminación de cookies desde JavaScript.....	168
Comprobación de cookies habilitadas.....	169
Prácticas del capítulo 9.....	169
 Capítulo 10:	
Objeto Image.....	171
Objeto Image.....	171
Propiedades del objeto Image.....	172

Precarga y visualización dinámica de imágenes.....172
Prácticas del capítulo 10.....174

Capítulo 11:

Gestión de eventos.....175
Programación de eventos.....175
 Eventos disponibles.....176
 Gestión del objeto evento.....178
 Gestión de errores.....180
 Consola de errores.....182
Prácticas del capítulo 11.....183

Capítulo 12:

Gestión del teclado y del ratón.....185
Gestión del ratón.....185
 Mensajes informativos.....186
 Mensaje informativo con respuesta.....188
 Detección del clic secundario del ratón189
Gestión del teclado.....190
 Mensaje de entrada de datos (método prompt()).....192
Reproducción de Sonido.....194
Prácticas del capítulo 12.....196

Capítulo 13:

Formularios.....197
Objeto form.....198
 Métodos del objeto form.....199
 Propiedades del objeto form.....199
 Eventos del objeto form.....199
Componentes del formulario HTML.....202
 Objeto input text.....202
 Objeto input button.....204
 Objeto input submit.....205
 Objeto input hidden.....206

Objeto input password.....	207
Objeto input radio.....	208
Atributo disabled.....	211
Objeto input checkbox.....	212
Objeto input file.....	215
Objeto textarea.....	216
Objeto select.....	217
Control de visibilidad por código	220
Seguridad en la entrada de datos.....	221
Segunda línea de protección.....	222
Riesgos clásicos.....	222
Los riesgos del código abierto.....	223
Tipos de ataques a la seguridad.....	223
Prácticas del capítulo 13.....	224
 Capítulo 14:	
DHTML y objeto document.....	225
DOM: modelo de objeto documento.....	226
Análisis del modelo DOM	226
DHTML y JavaScript.....	230
Objeto document.....	230
Método getElementById().....	231
Principales métodos del objeto document.....	231
Contenido HTML de un elemento	231
Clase style.....	231
Propiedad className.....	232
Propiedades de estilo.....	235
Prácticas del capítulo 14.....	255
 Capítulo 15:	
DOM.....	257
DOM: Modelo de objeto Documento	257
DOM o BOM.....	258
Estructura de un documento HTML.....	258

Objetos DOM.....	259
Tipos de nodos.....	260
Interpretación del modelo	260
Node.....	262
NodeList.....	267
NamedNodeMap.....	267
Document.....	268
Element.....	270
Attr.....	272
CharacterData.....	272
Text.....	273
Comment.....	273
CDATASection.....	273
DocumentType.....	273
Notation.....	274
Entity.....	274
EntityReference.....	274
Ejemplos.....	275
Búsqueda de un nodo.....	275
Referencia al elemento de raíz.....	275
Navegación por el documento.....	276
Añadido de nuevos elementos a la estructura.....	277
Asignación de atributos a un elemento.....	278
Eventos en el modelo DOM.....	278
Gestión de tablas con DOM.....	280
Prácticas del capítulo 15.....	284

Capítulo 16:

Ajax y JavaScript.....	287
¿Qué resuelve Ajax?.....	287
Composición de tecnologías	288
Modelo clásico de una aplicación Web.....	289
Modelo Ajax de una aplicación Web.....	290
¿Qué se obtiene con Ajax?.....	292
¿Qué se requiere para usar Ajax?.....	293

Ajax en acción.....	294
Las tecnologías relacionadas con Ajax.....	294
XML, XSLT y XPath.....	294
Modelo de objeto Documento (DOM).....	296
Las limitaciones de Ajax.....	297
Programación Ajax.....	298
Método open().....	299
Método send().....	300
Otros métodos del objeto XMLHttpRequest.....	301
Programación asincrónica.....	301
Prácticas del capítulo 16.....	303
Apéndice A: Respuestas.....	305
Índice.....	321

Vistazo parcial del libro
Salto de páginas...

Capítulo 2: Conceptos básicos

Contenido

- ◆ ***Estructura del código***
- ◆ ***Normas básicas de codificación***
- ◆ ***Tipos de variables***
- ◆ ***Sentencias condicionales***
- ◆ ***Control de bucles***
- ◆ ***Funciones***
- ◆ ***Ámbitos de variables***

Antes de comenzar analizar los conceptos de la programación JavaScript es preciso que el lector conozca ciertas reglas básicas de la codificación que en realidad son aplicables a todos los lenguajes en general, como Java, PHP, C++, etc.

Si el lector ya tiene experiencia en algún lenguaje quizá ya conozca estas normas básicas y esté habituado a aplicarlas.

Cuando se comienza el camino para llegar a ser un programador profesional no se suele tener conciencia de la importancia de que nuestro código sea limpio, claro y conciso; en un principio el aprendiz suele estar más que contento con que su código funcione y que además haga lo que se pretende que haga. Con el correr del tiempo y con la experiencia de tener que mantener código programado por terceros nos vamos dando cuenta que los

programas más simples podrían llegar a ser totalmente ilegibles si no se respeta una serie de criterios de organización y estructura. Es cierto que los programas mal estructurados pueden funcionar perfectamente y quizás con un tiempo de respuesta excelente, pero la realidad es que esos programas duran muy poco tiempo, dado que al primer mantenimiento (siempre habrá necesidad de modificar un código) ese programa normalmente se tiene que reprogramar en su totalidad porque casi seguro que ésa es la solución más rápida. Y si un programa dura poco es antieconómico.

Puede parecer exagerado decir que "siempre" habrá necesidad de modificar un código y la realidad no es en absoluto una exageración. Durante su vida, un programa sufre dos clases de tareas de mantenimiento:

- **Corrección de errores:** Eliminación de fallos detectados al presentarse situaciones no debidamente probadas durante la fase del desarrollo.
- **Mantenimiento evolutivo:** A lo largo del tiempo un programa se modifica debido a optimizaciones, cambios de diseño, inclusión de nuevas funcionalidades, adaptaciones para nuevas versiones de navegadores, etc.

Con un trabajo profundo y completo en las pruebas del programa podemos llegar a reducir a 0 la corrección de errores en tiempo de producción; es decir, podemos llegar a poner en producción un software sin errores (también se precisa tiempo y algo de suerte porque nuestro código no vive sólo en una burbuja sino que se integra con otros componentes que pueden cambiar a lo largo del tiempo y generar respuestas no previstas en el momento de la creación original de nuestro programa); pero nadie nos protege del mantenimiento evolutivo (nuestro cliente quiere un retoque en el diseño de la interfaz, a nosotros se nos ocurre que una acción determinada se puede gestionar de otra forma más impactante al usuario, se debe añadir un enlace a otra página, desaparece un patrocinador y debemos modificar la rutina de anuncios, etc. y podríamos seguir enumerando cientos de motivos de "mantenimiento evolutivo").

Entonces ha quedado clara la idea de que un programa no es algo que se escribe una vez y que después nos podemos olvidar de él para siempre. Todo lo contrario, lo normal es que volveremos sobre sus líneas una y otra vez. Sabiendo esto debemos tomar en consideración que mantener un programa claro, bien estructurado y que cumple con ciertas normas de nomenclatura requiere mucho menos tiempo de comprensión del código, es mucho más reutilizable y es menos propenso a los errores. Dicho esto, el lector entenderá que aplicar las siguientes normas o consejos no es ningún castigo sino una inversión a futuro.

Estructura del código

Un modo para incrementar la legibilidad del código fuente es utilizar normas de sangría para que el código quede alineado de modo lógico separando de modo visual los bloques de sentencias.

La sangría del código es la norma más simple y efectiva para que el código resulte más fácil de comprender mediante el uso de tabuladores para señalar la dependencia de los bloques lógicos.

A continuación se codifica el mismo bloque de código de tres modos diferentes y el lector verá fácilmente las ventajas del uso de la sangría:

```
if ( colnum==tope) {
    colnum = colnum +2;
} else {
    colnum= 0;
}
```

O la alternativa 1:

```
if ( colnum==tope) {colnum = colnum +2;} else {colnum= 0;}
```

O la alternativa 2:

```
if ( colnum==tope) {
colnum = colnum +2;
} else {
colnum= 0;
}
```

Pese a que el ejemplo es muy sencillo es fácil visualizar la ventaja del uso de una sangría adecuada. La sangría no implica que el código se ejecutará más rápido, el tiempo de ejecución es exactamente el mismo en los tres casos. El beneficio está en la legibilidad y en la comprensión del código en las tareas de mantenimiento.

Los editores de texto nos permiten configurar la opción de sangría o tabulación (también denominado indentado del código) de modo que se visualizan los espacios y los caracteres de tabulación.

Comentarios en el código

JavaScript no es una excepción y como casi todos los lenguajes permite el uso de líneas de comentarios dentro del propio código fuente. Los comentarios bien colocados, concisos y precisos son una bendición para el programador que debe realizar un mantenimiento sobre un código realizado por otro; no es cuestión de escribir una novela antes de cada sentencia ya que ése es un defecto que debemos evitar, sino de saber explicar el objetivo de cada bloque de código que no quede claro a primera vista, debemos tener en cuenta que las sentencias escritas en JavaScript pueden ser autoexplicativas cuando se utiliza una correcta nomenclatura de variables (que trataremos más adelante) y cuando se utilizan las instrucciones esperadas dentro del punto de vista lógico.

Los comentarios son líneas no ejecutables por lo que su inclusión no incrementa el tiempo de ejecución del código.

Los comentarios JavaScript se puede especificar de tres modos diferentes:

```
// Esto es un comentario
num = 5;

/* esto
es
un
comentario */
num = 5;

num = 5; // Esto es un comentario
```

La forma `/* ...*/` permite que un comentario se extienda por más de una línea hasta llegar al indicador de cierre `*/`.

Como regla general un comentario debe ser concreto y conciso con un texto que explique el objetivo del bloque de código siguiente y no debe explicar lo que resulta obvio por la simple lectura de la instrucción.

Si bien los comentarios no se ejecutan en el navegador, los comentarios viajan hasta el navegador como parte del archivo JavaScript y el cliente los puede visualizar con la opción Ver código fuente (o como se denomine en los distintos navegadores) por lo que no se debe incluir información que afecte la seguridad del sitio (contraseñas, cadena de conexión, etc.).

Cuando se realiza una modificación en un programa es aconsejable incluir un comentario que indique la fecha de la modificación, en el futuro puede ser un dato que nos aporte una información que nos ayude en la tarea de mantenimiento.

Nomenclatura

No existen reglas universales y sagradas para la nomenclatura de los objetos o entidades de un programa; si el lector ha trabajado en más de un sitio de desarrollo de software habrá podido comprobar que no existe una norma fija pero lo que sí existe (normalmente) es "alguna" norma de nomenclatura que todo el equipo de desarrollo debe seguir. Lo peor de todo es dejar al libre albedrío la elección de normas personales o individuales.

Cuando el desarrollador trabaja solo tiene la libertad de elección pero lo adecuado es que adopte un criterio determinado y que lo mantenga estrictamente. También se beneficiará de esa metodología de trabajo cuando tenga que leer sus propios programas meses después de su programación original. Además, aunque posiblemente comencemos trabajando solos, si las cosas nos van bien en el futuro quizá tengamos un colaborador o más que tendrán la tarea de mantener nuestro código.

La principal regla de la nomenclatura es mantener la coherencia y un esquema de formato en el nombre de cada cosa (variable, función, objeto, módulo, etc.). La regla puede ser personal pero si es coherente cualquier programador puede entenderla rápidamente después de leer un poco el código de nuestro programa.

Nomenclatura de variables

La primera regla en la nomenclatura de las variables es que se debe intentar que su nombre tenga relación con su contenido. La separación entre palabras se indica con una mayúscula y se suele utilizar un prefijo que describe el tipo de dato, por ejemplo:

- **txtLíneaDetalle**: Define que la variable es una cadena de texto (prefijo txt) y que contendrá una línea de detalle.
- **dtFechaEntrada**: Define que la variable es de tipo fecha y hora (prefijo dt) y que contendrá la fecha de entrada.
- **intContadorLíneas**: Define que la variable es de tipo entero (prefijo int) y que contendrá un contador de líneas.
- **bRetorno**: Define que la variable es de tipo booleano (prefijo b) y que contendrá el valor de retorno.

Los prefijos que indican los tipos de las variables son de libre elección aunque los citados en estos ejemplos son muy utilizados. También se suele utilizar str para indicar una cadena de caracteres, n para un tipo entero, etc.

Lo importante es ser coherente a lo largo de la aplicación y no inventar prefijos que no se expliquen por sí solos.

Nomenclatura de funciones

El criterio de nomenclatura de las funciones es similar al que hemos visto en las variables. La principal diferencia es que el prefijo suele indicar la acción principal de la función; el prefijo puede escribirse en inglés (es más conciso que el español) o en el idioma que más se prefiera, lo importante es nuevamente ser coherente a lo largo de todo el desarrollo:

- **get**: Obtiene un valor. Por ejemplo, `getNro()`, `obtenerNro()`.
- **set**: Asigna un valor. Por ejemplo, `setNro()`, `asignarNro()`.
- **is**: Devuelve verdadero o falso. Por ejemplo, `isPar()`, `esPar()`.
- **add**: Añade un elemento. Por ejemplo, `addElemento()`, `agregarElemento()`.
- **print**: Imprime un objeto. Por ejemplo, `printFormulario()`, `imprimirFormulario()`.

A continuación del prefijo se añade la descripción de la función en una o más palabras, utilizando una mayúscula para indicar el comienzo de cada palabra. Por ejemplo:

- **isObjetoEmpleado()**: Este nombre de función es adecuado para saber si un objeto es de tipo Empleado.
- **getColor()**: Este nombre de función es adecuado para obtener el color de un objeto.

Longitud de los nombres

Por razones de legibilidad no son adecuados los nombres demasiado cortos (sin significado aparente) y los nombres excesivamente largos (más de 16 o 18 caracteres) por el espacio que ocupan en la línea de edición.

La excepción a esta recomendación es el típico uso de las variables utilizadas como índices (en bucles o en elementos de matrices), en estos casos es habitual utilizar la variable `i` o `ix` para dar a entender que es un índice y también `x` e `y` para indicar coordenadas de la pantalla.

Por norma general se acepta un nombre de variable con una o dos letras si es una variable transitoria que se utiliza y se destruye en pocas líneas.

Palabras reservadas

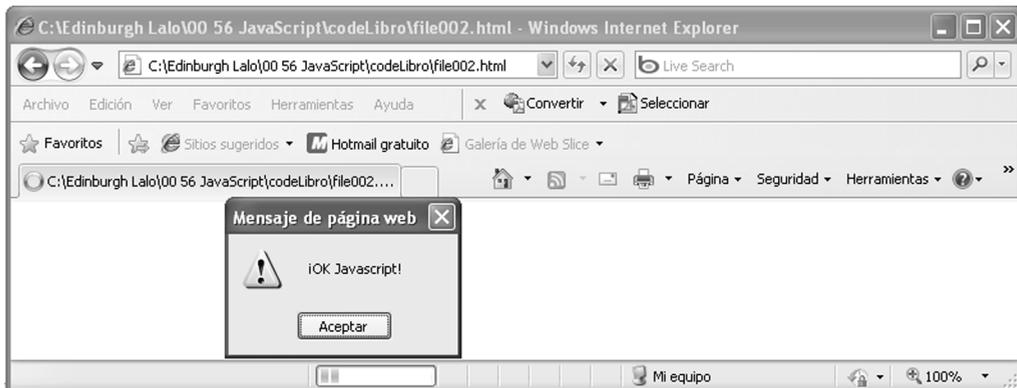
Como en todos los lenguajes de programación, el nombre que se asigne a variables y funciones tiene la habitual restricción que impide el uso de las palabras reservadas del lenguaje, básicamente, el nombre de las sentencias propias de JavaScript.

Estas recomendaciones son generales y se aplican a todos los lenguajes de programación.

¿Está activado JavaScript?

Para verificar si JavaScript está activado en un navegador podemos utilizar este código:

```
<SCRIPT language=javascript>
alert("¡OK Javascript!");
</SCRIPT>
<NOSCRIPT>
No está activado JavaScript.
</NOSCRIPT>
```



Scripts de JavaScript

El ámbito habitual del código JavaScript es dentro de una página HTML. Seguramente el lector ya conoce el código HTML, sus etiquetas y sus posibilidades pero lo que aún quizá no sepa es cómo se integra el código JavaScript dentro de una página HTML.

Dentro de un documento HTML se indica la inclusión de código JavaScript mediante alguno de estos métodos:

```
...  
<script type="text/javascript" src="[xxx]"></script>  
...
```

Siendo [xxx] la dirección URL relativa o absoluta del recurso con código JavaScript, el cual tendrá extensión js. Este método es más seguro y recomendado.

Otro modo sería incluir código directamente en el documento entre los elementos `<script>` y `</script>`:

```
...  
<script type="text/javascript">  
<!--  
    // código JavaScript  
-->  
</script>  
...
```

Dentro de un documento HTML, las etiquetas `<script>` `</script>` marcan el inicio y el final de un área de código.

Modo indirecto

También es posible integrar código JavaScript mediante una referencia a un archivo externo de extensión js que contiene las funciones utilizadas en la página:

```
<script type="type/javascript" src="miarchivo.js"></script>
```

El atributo `src` indica al navegador que el código fuente JavaScript está dentro del archivo `miarchivo.js`. Desde el punto de vista funcional esto es lo mismo a haber colocado el código JavaScript dentro de la página.

El uso de archivos no sólo es más recomendable que el uso del código directo por motivos de organización, mantenimiento, limpieza del código y seguridad sino que también representa ciertas ventajas de rendimiento. Cuando el navegador utiliza caché coloca copias de los documentos descargados desde Internet durante cierto tiempo; cuando el navegador detecta que una referencia la tiene en caché no la descarga desde el servidor sino que utiliza la copia local.

La ventaja del uso de caché tiene un detalle que se debe tener en cuenta cuando en pruebas modificamos el archivo .js. Para que el navegador utilice la versión actualizada debemos asegurarnos que se actualiza también el área de caché. Por ejemplo, en Internet Explorer se utiliza la ficha General accesible desde Herramientas/Opciones de Internet para acceder al cuadro de diálogo que nos permite eliminar, entre otras cosas, las páginas temporales de Internet.



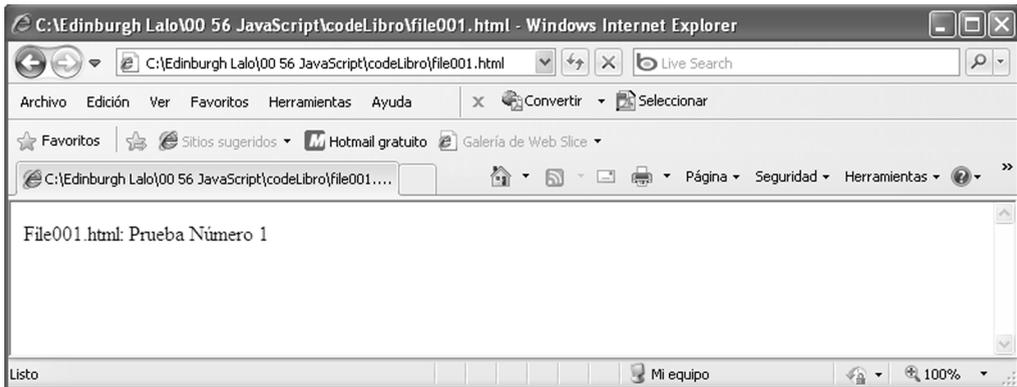
A continuación codificaremos nuestro primer script que nos servirá para comprobar que nuestro navegador está configurado para procesar código JavaScript.

```
<script type="text/javascript">
    document.write("File001.html: Prueba Número 1");
</script>
```

Para abrir este archivo en el navegador podemos utilizar la dirección física del archivo utilizando alguna de estas codificaciones adaptando la ruta de acceso a la que corresponda a su equipo:

```
D:\Test\Code\file001.html
file:///D:/Test/Code/file001.html
```

Al abrir el archivo en un navegador, por ejemplo, IE obtendríamos esta página:



Si no se visualiza el texto generado por JavaScript lo más probable es que sea porque en su navegador esté desactivada la opción JavaScript.

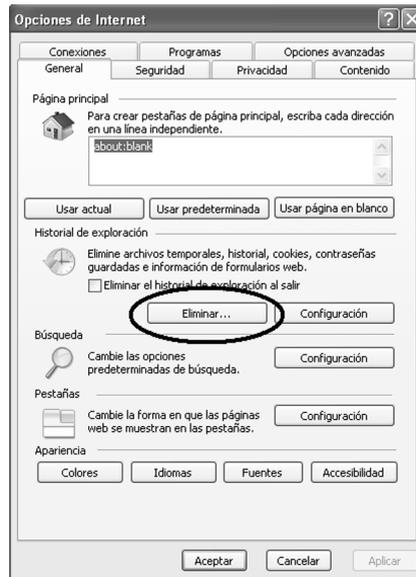
Uso de archivos externos

La opción más recomendada para el uso de código JavaScript es mediante la referencia a un archivo externo, además de tener algunas ventajas de seguridad cuando se trata de un código al que se hace referencia en varias páginas de nuestro sitio también se puede ganar en rendimiento dado que el navegador suele almacenar en caché los archivos que se hayan referenciado recientemente por lo que si el archivo está almacenado en caché (es decir, en el disco duro local del cliente) el acceso y la descarga son inmediatos sin necesidad de ir hasta el servidor web. De esta manera las páginas se aligeran de peso (menos KB) y se cargan más rápidamente (salvo la primera que exige descargar el archivo js).

El uso de archivos externos también resulta beneficioso cuando llega el momento de realizar mantenimientos dado que el código está centralizado y no repetido en varias páginas.

El uso del caché tiene ventajas de rendimiento pero muchas veces obtenemos resultados no esperados cuando después de realizar cambios en el código vemos que la página sigue saliendo como si no hubiéramos cambiado nada. La razón es que el almacenamiento de caché sigue manteniendo la versión antigua del código: la solución vaciar el caché y volver a intentar la prueba. Con la zona de caché vacía, el navegador debe ir a buscar todo al servidor web y allí encontrará la nueva versión del código.

El modo de limpiar la zona de caché de archivos temporales varía entre los distintos navegadores, por ejemplo, en IE se accede al menú Herramientas/Opciones de Internet y en la ficha General se pulsa el botón Eliminar archivos.



Bloque de instrucciones

Cada línea de código finaliza con punto y coma y los bloques de instrucciones se encierran entre llaves.

```
<SCRIPT type="text/javascript">
  var var1 = 5;
  document.write("Hola JavaScript<BR \>");
  if (var1 > 2) {
    document.write("La variable var1 es igual a:<BR \>");
    document.write(var1);
  }
</SCRIPT>
```

Las etiquetas `<noscript>` no son obligatorias pero es probable que las encontremos en alguna página; estas etiquetas se presuponen después de cada cierre de la etiqueta `</script>` y por lo tanto no son necesarias. El contenido encerrado por las etiquetas `<noscript>` y `</noscript>` es código HTML.

Variables

Todos los lenguajes de programación utilizan variables para almacenar y gestionar la información que necesita el programa para implementar su lógica. Básicamente, una variable es un área de memoria que contiene un dato al que se puede acceder por medio de nombre.

Vistazo parcial del libro
Salto de páginas...

Capítulo 9: Cookies

Contenido

- ◆ *Archivos cookies y la seguridad*
- ◆ *Limitaciones de las cookies*
- ◆ *Propiedad cookie del objeto document*
- ◆ *Escritura de cookie*
- ◆ *Lectura de cookies*
- ◆ *Eliminación*

Los archivos cookies representan una de las soluciones más simples para almacenar datos que facilitan la navegación por un sitio web en el disco duro del equipo del cliente. Los datos que se almacenan se reutilizan en las siguientes visitas al sitio. Lo que se puede guardar en un archivo cookie depende totalmente del diseñador del sitio, pero en general se suele utilizar para

los siguientes fines:

- Almacenar identificadores para el acceso a un sitio.
- Almacenar datos relacionados con las preferencias seleccionadas por el usuario al navegar por un sitio.
- Almacenar datos de un proceso de navegación inconcluso (por ejemplo, la cesta de compra no terminada) .
- Almacenar datos de la última visita a un sitio (por ejemplo, el último artículo leído en un foro).

Los archivos cookies son archivos de texto simple; el sitio en donde se graban estos archivos varía según el navegador pero la funcionalidad de estos archivos es siempre la misma. Aunque el navegador puede registrar varios archivos cookies por sitio tiene un límite de 3 KB por dominio.

Sabemos que JavaScript tiene prohibido leer o escribir en el disco del cliente pero la propiedad cookie representa el único medio por la que se pueden hacer estas operaciones mediante el navegador.

Un archivo cookie se define mediante los datos siguientes:

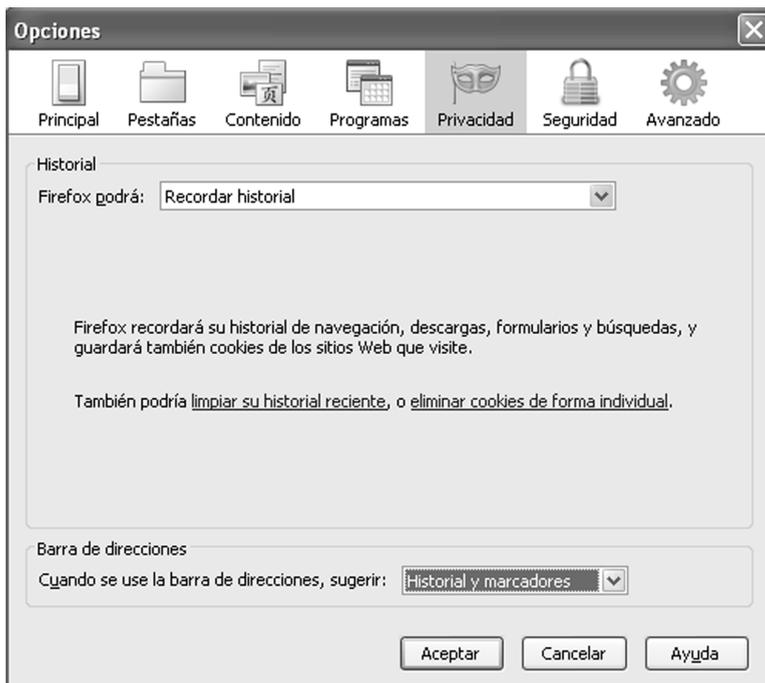
- Nombre del archivo cookie para su identificación
- Un valor asociado al nombre
- Una fecha de caducidad
- Un dominio para el que el archivo cookie tiene validez
- Una ruta que define el acceso de la página para la que el archivo cookie es legible
- Un indicador de seguridad que hace que el archivo cookie sea inaccesible desde páginas no seguras, es decir, que no utilizan el protocolo https.

Cookies y seguridad

Los archivos cookies tienen una excesiva mala fama entre los usuarios lo que ha llevado a que los navegadores incluyan siempre la opción de rechazar el uso de cookies. La realidad es que la funcionalidad de los archivos cookies no debería generar tantas preocupaciones. Esto tiene su efecto en el diseño de los sitios web ya que los desarrolladores no deberían hacer depender las funcionalidades de los sitios en modo exclusivo en el uso de cookies, una decisión así podría hacernos perder muchos visitantes y eso es algo que nadie desea.

Los navegadores son los que gestionan los archivos cookies y el usuario es quien configura las opciones.

Por ejemplo, Firefox incluye el siguiente cuadro de configuración para el uso de cookies:



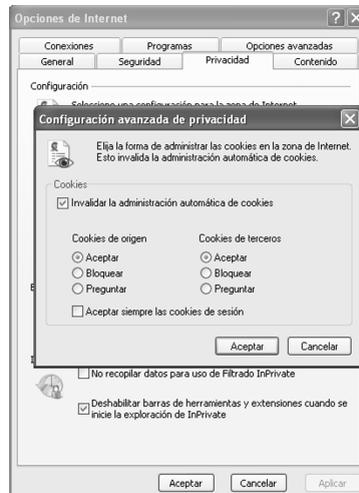
A este cuadro de diálogo se accede desde Herramientas/Opciones en la ficha Privacidad. Desde este cuadro se puede consultar el estado de los archivos cookies.



Desde este cuadro podemos visualizar los archivos cookie por dominio y por nombre; opcionalmente podemos eliminar un archivo cookie o todos los archivos cookie.

El contenido del archivo cookie es totalmente visualizable y también se puede modificar y eliminar con facilidad.

En Internet Explorer podemos utilizar Herramientas/Opciones de Internet para acceder a la pestaña Seguridad, donde encontraremos el botón Avanzada para acceder al cuadro de configuración que nos permite fijar la política de bloqueo de cookies (de origen y de terceros).



Limitaciones de las cookies

La principal limitación en el uso de cookies, y funcionalmente mucho más importante que su límite de tamaño, es que nunca podemos estar seguros que encontraremos un archivo cookie escrito en una sesión anterior. Ya hemos visto en la sección anterior que el usuario puede borrarlos en cualquier momento, cambiar la configuración del navegador, cambiar de navegador, cambiar de PC, etc., por lo que nunca estaremos seguros de obtener los datos registrados. Esto hace que todas las estadísticas que se basen en el uso de cookies tendrán un margen de error importante y difícil de estimar.

La segunda limitación es más fácil de controlar dado que las reglas son claras: el espacio máximo asignado para un dominio es aproximadamente de 3 KB. Dado que cada archivo cookie tiene una serie de datos fijos (fecha de

caducidad, dominio, nombre, etc.) es preferible utilizar la menor cantidad posible de nombres y de sea manera aprovechar al máximo los 3 KB asignados.

Interacción con otros lenguajes

Un dato cookie creado por JavaScript se puede leer desde un lenguaje del servidor, por ejemplo, PHP. Del mismo modo también se cumple lo inverso: un archivo cookie creado con PHP se puede leer con código JavaScript. Esto abre otra posibilidad de intercomunicación entre el lenguaje cliente y el lenguaje del servidor.

Gestión de cookies

JavaScript no implementa métodos básicos para la gestión de cookies por lo que nos obliga a definir funciones personalizadas para resolver ese vacío.

Escritura de cookies

Normalmente se crea una función con el nombre `setCookie()` aunque la función podría tener cualquiera de los nombres válidos para funciones JavaScript. En este caso haremos un ejemplo con `asignarCookie()`.

Al crear un archivo cookie tenemos varios parámetros para asignar pero sólo dos son obligatorios: nombre y valor.

Una función de uso genérico podría tener este contenido:

```
function asignarCookie(nombre, valor, caducidad, ruta, dominio,
    seguridad) {

document.cookie=nombre+"="+escape(valor)+
((caducidad ==undefined) ? "" : ("; expires="+
caducidad.toGMTString()))+
((ruta==undefined) ? "" : ("; path="+ruta))+
((dominio==undefined) ? "" : ("; domain="+dominio))+
((seguridad==true) ? "; secure" : "");

}
```

El dato cookie se almacena en la propiedad `cookie` del objeto `document`. Se asigna con una cadena de caracteres que incluye nombre, valor y una serie de parámetros opcionales.

cookie es una propiedad del objeto document.

En la función se emplea repetidamente el operador ternario para gestionar los casos de indefinición de alguno de los parámetros; por ejemplo:

```
(caducidad ==undefined) ? "" : ("  
expires="+caducidad.toGMTString())
```

Si el parámetro `caducidad` estuviese indefinido se asignaría una cadena vacía y si estuviese informado se asignaría el parámetro `expires`. Lo mismo se realiza en otros parámetros.

- **nombre:** Es el nombre de cookie. Se aceptan letras y números.
- **valor:** Es el valor que se almacena en el archivo; para gestionar correctamente ciertos caracteres especiales se utiliza la función `escape()` para que realice su codificación. Esto obligará el uso de la función `unescape()`.
- **caducidad:** Se utiliza para informar el parámetro `expires`. Es un objeto `Date`. Si no se informa, el dato cookie se eliminará al finalizar la sesión vigente. La sesión finaliza al cerrar todas las ventanas del navegador y en ese momento se eliminan todos los datos cookies que no tienen fecha de caducidad.
- **ruta:** Se utiliza para informar el parámetro `path`. Es el camino de acceso que corresponde al dato cookie. Para autorizar el dato cookie para todas las carpetas del sitio se debe informar `"/`. Si no se informa, se supone que el archivo cookie es válido sólo para esa página.
- **dominio:** Se utiliza para informar el parámetro `domain`. Permite definir el dominio del cookie.
- **seguridad:** Se utiliza para informar el parámetro `secure`. El valor `true` corresponde a conexiones seguras de `https`.

Para usar esta función que hemos definido podemos utilizar esta sintaxis:

```
var fecha = new Date();  
// se asigna la fecha de hoy más 30 días  
  
fecha.setTime(fecha.getTime()+30*24*3600*100);  
  
asignarCookie("LibrosInfo1", "datos xxxx", fecha, "/", "dom.com");  
asignarCookie("LibrosInfo2", "datos yyyy", fecha, "/", "dom.com");
```

Lectura de cookies

De igual modo a la escritura de cookies, JavaScript tampoco implementa una función o un método para la lectura de cookies. Debemos realizarlo mediante código propio.

Para leer el contenido se analiza la propiedad `cookie` del objeto `document` con el nombre de `cookie` que se haya asignado. La propiedad `document.cookie` almacena todos los datos de `cookies` y en realidad lo habitual es que nos interese obtener sólo un `cookie` determinado por lo que tendremos que trabajar para obtener la parte de la cadena que nos interesa.

Para que el lector sepa a qué nos enfrentaremos, éste es el formato de una propiedad `cookie` de ejemplo :

```
nombre1=valor1; nombre2=valor2; ...nombreN=valorN
```

Tal como se puede observar el contenido de la propiedad no coincide con el formato utilizado en la asignación, ya que en lectura sólo se tiene acceso a la lista de nombres y valores. Entre `cookie` y `cookie` se utiliza `";` como separador.

Veamos una propuesta de función para el análisis de la propiedad `cookie`, a la función la denominaremos `leerCookie()` que recibirá un único parámetro: el nombre de `cookie` cuyo valor se desea obtener:

```
function leerCookie(nombre) {
// si la propiedad no tiene información se
// devuelve null
if (document.cookie.length== 0) {
    // no hay cookie registrado
    return null;
}
// separación de la cadena en grupos
// clave=valor
var sep1 = new RegExp("; ", "g");
var listaCookies = document.cookie.split(sep1);
for (var i = 0; i < listaCookies.length ; i++) {
    // separar la parte de clave de
    // la parte valor (separador =)
    var sep2 = new RegExp("=", "g");
    var cookie = listaCookies[i].split(sep2);
    // se compara el nombre de cookie
    // con el nombre buscado
    if (cookie[0] == nombre) {
        // se aplica la función unescape
        // para reponer los caracteres
        // originales
        return unescape(cookie[1]);
    }
    // si no encontró el nombre buscado
    // devuelve null
    return null;
}
}
```

La función leerCookie() devuelve null en dos casos: si la propiedad cookie está vacía o si no existe el nombre de cookie buscado.

Web Developer y cookies

Podemos complementar este ejemplo utilizando la barra de herramientas Web Developer del navegador Firefox que nos permite acceder a los datos cookie de la página. Al hacer clic en Ver información de las cookies dentro del menú Cookies aparece la ficha con todas las propiedades de las cookies registradas para el dominio.

Para cada cookie se indican los parámetros de su definición:

Información de las cookies - http://www.continental.com.ar

/player.aspx?id=zfE%2bcCh18sA0IUhnMVp0qTskJ%2b%2f0iAG%2f1HGkEHsn%2fsZJwaFvDOirpVxg5Fz8S5uDQ5IAFLS7ED6lqWxxTDRs75Vnd0oUpmQZ%3d%3d

3 cookies

NOMBRE	VALOR	HOST	RUTA	SEGURIDAD	CADUCA
eUser	nouser	.continental.com.ar	/	No	Sat, 15 Jan 2011 10:34:03 GMT

Ejemplo

Ya tenemos definidas las dos funciones para la gestión de los archivos cookie por lo que estamos en condiciones de desarrollar una página que las utilice en conjunto:

```
<html>
<head><title>Ejemplo del uso de cookies </title>
<script type="text/JavaScript">

function leerCookie(nombre) {
```

Vistazo parcial del libro
Salto de páginas...

Apéndice A: Respuestas

Respuestas del capítulo 1

1. Nada. JavaScript está integrado en todos los navegadores actuales (IE, Firefox, Opera, Chrome, Netscape, etc.). Lo que sí puede ser necesario es simplemente activar la opción del uso de JavaScript dentro del navegador.
2. No, no se puede ni leer ni escribir en una base de datos directamente desde código JavaScript; estas acciones se deben realizar mediante un lenguaje del servidor, por ejemplo, ASP o PHP.
3. No, está prohibida la lectura y escrituras de archivos de todo tipo desde código JavaScript. Si se quiere realizar este tipo de operaciones en el disco del servidor se debe apelar a uno de los lenguajes del servidor, por ejemplo, ASP o PHP. Si se quiere realizar este tipo de operaciones en el cliente se debería utilizar código Java o funciones ActiveX, pero son acciones que el cliente debe autorizar.
4. Falso.
5. Verdadero.

6. Falso. Existen casos en que la compatibilidad puede ser total, por lo que la respuesta Verdadero en esos casos sería correcta, pero la respuesta correcta es Falso porque las implementaciones de JavaScript son diferentes entre los distintos navegadores, por lo que es probable que tengamos que programar algunas instrucciones específicas para las distintas versiones.
7. Falso.

Respuestas del capítulo 2

1. Falso.
2. Con la etiqueta `<script>` y `</script>`
3. Falso.
4. Indefinido, número, cadena y booleano.
5. Hay una conversión automática y el resultado es 400, como si la variable `var1` fuese de tipo `number`.
6. Esta función es adecuada para comparar una variable en una sentencia `if`:

```
if (var1 = 2) {  
  ...  
}
```

7. 6 iteraciones.
8. Verdadero.
9. Mediante un archivo de texto con la extensión `.js` que posteriormente se incluye en la página HTML con esta sintaxis:

```
<SCRIPT language="javascript" src="miArchivo.js"></SCRIPT>
```

10. Mediante este código se sabe inmediatamente cómo es el estado del navegador respecto a la gestión de JavaScript:

```
<SCRIPT language=javascript>  
alert(";OK Javascript!");  
</SCRIPT>  
<NOSCRIPT>  
No está activado JavaScript.  
</NOSCRIPT>
```

11. Correcciones de errores y mantenimiento evolutivo.
 12. if, if/else y switch
 13. for, while y do
 14. Las condiciones que se evalúan pueden ser simples o complejas y, en este caso, se unen mediante los operadores lógicos && y || (respectivamente, Y - O)
 15. No.
 16. Sí, pero deben ser los últimos de la lista de parámetros.
 17. Sí, una variable puede contener un tipo de dato en un momento y después se le puede asignar cualquier otro tipo de datos.
 - 18.
- ```
iLínea = iLínea < 60 ? iLínea+ : 0;
```
19. No, pero es recomendable.
  20. En una línea se puede utilizar //:

```
// esto es un comentario
```

El modo /\* ...\*/ permite comentarios de varias líneas:

```
/* Esto
```

```
es
un comentario */
```

## Respuestas del capítulo 3

1. Falso, los conceptos más complejos de la teoría no están implementados pero de todas maneras se puede considerar un lenguaje orientado a objetos dado que básicamente gestiona objetos de tipo intrínseco y personalizados.
2. Es una entidad que reúne datos y funciones para gestionar esos datos. En terminología POO son propiedades y métodos.
3. Verdadero. Aunque el término método se suele reservar para las funciones relacionadas a un objeto determinado. Las funciones auxiliares de un programa se denominan simplemente funciones aunque si somos estrictos y consideramos al programa como un objeto, tampoco es

incorrecto denominar métodos a esas funciones auxiliares (aunque no es lo habitual).

4. Se utiliza la sintaxis del punto:

```
nombreObjeto.nombrePropiedad
```

5. Cada objeto debe tener su constructor en el que se crea su instancia en memoria y opcionalmente se inicializan sus propiedades.
6. No es obligatorio que un objeto tenga propiedades o métodos (salvo el constructor), pero lo lógico es que al menos tenga alguna de las dos cosas. El método constructor es obligatorio.
7. Utilizando el operador `new`.
8. La variable `miEquipo2` se crea correctamente. Potencialmente podría haber un problema si se piensa que las dos variables son diferentes. Las dos variables ocupan la misma memoria por lo que no son independientes entre sí.
9. Dentro del código relacionado a un objeto, la palabra clave `this` representa al propio objeto.
10. Un objeto `A` puede tener propiedades de distintos tipos (números, cadenas, fechas, objetos personalizados de otras clases, etc.). Estos objetos están contenidos dentro del objeto `A`.
11. Se define un nombre de método al que se le asigna una función (por ejemplo, `fnImprimir()`) que se desarrolla aparte.

```
...
// definición de métodos
this.imprimir = fnImprimir;
...
```

12. Son dos objetos que tienen dos nombres diferentes pero que comparten la memoria, por ejemplo, al modificar uno también se modifica el otro.

## Respuestas del capítulo 4

1. Es un conjunto de datos a los que se accede con un mismo nombre de variable y utilizando un índice que puede ser numérico o una clave.
2. El objeto `Array`.
3. No hay un límite físico pero desde el punto de vista lógico habitualmente se utilizan matrices entre 1 y 3 dimensiones.

4. El primer elemento tiene índice 0, el último lo da la propiedad length - 1. Siendo length la cantidad total de elementos.
5. La sentencia for.
6. La sentencia for...in
- 7.

```
<SCRIPT language=javascript>
function listarDiagonal(mat) {
 for (var fila = 0; fila <mat.length; fila++){
 if (mat[fila][fila] != undefined) {
 document.write("Elemento mat[" + fila + "][" +
fila, "] ",
 mat[fila][fila], "
");
 }
 }
}

var mat = new Array();
mat[0] = new Array(1, 4, 5);
mat[1] = new Array(8, 6, 0);
mat[2] = new Array(1, 5, 8);
mat[3] = new Array(2, 7, 5);
listarDiagonal(mat);
</SCRIPT>
```

8. 1, 1, 1, 2, 2, 2
9. El método joint() convierte una matriz en una cadena de caracteres formada por cada uno de los elementos de la matriz separados por un carácter que se define como parámetro del método.
10. Una matriz con este contenido: 1, 2, 3.

## Respuestas del capítulo 5

1. Realmente no; pero sí se podría optimizar el objeto Date proporcionado por JavaScript incluyéndolo dentro de un objeto personalizado para así añadirle los métodos que nos hagan falta. De esta manera aprovechamos la implementación vigente del objeto Date y simplemente le añadimos lo que nos parezca que es necesario, por ejemplo, funciones especiales de formateo, diferencia de fechas teniendo en cuenta sólo los días comerciales, etc.
2. El milisegundo.
3. Se debe informar el número de mes real - 1 debido a que el dato se basa en 0, es decir, enero es 0.

4. La función devuelve el desfase en minutos entre la hora local y el horario del meridiano Greenwich. La hora local se toma del equipo del cliente (no del servidor de la página web).
5. Mediante el uso de la función `setTimeout()` o también `setInterval`.

## Respuestas del capítulo 6

1. Falso. Es un objeto `String` aunque no se haya creado con el operador `new`.
2. Simplemente concatenando una cadena vacía, la variable resultante será una cadena.

```
var miNro=5000.66;
var miCadena= miNro + ""; // ahora miCadena será "5000.66"
```

3. Con el operador `+` o con el método `concat()`.
4. Con el método `charAt()`.
5. Mediante el método `substring()`.
6. El método `slice()` es idéntico al método `substring()`. En el método `substring()` se indica la posición de inicio y de fin de la subcadena, mientras que el método `substr()` se indica la posición de inicio y la longitud.
7. `indexOf()`.
8. Funcionan todos de modo similar. Básicamente rodean una cadena con unas etiquetas HTML, que varían según el método utilizado. Por ejemplo, el método `anchor()` añade etiquetas `<A>` `</A>`.
9. JavaScript permite evaluar una cadena de caracteres como si se tratase de código fuente JavaScript. La función `eval()` ejecuta el código y retorna el resultado.
10. Representan un mecanismo que simplifica ciertas manipulaciones de las cadenas de texto, como verificaciones de formatos complejos (emails, número telefónicos, etc.), extracciones de subcadenas, etc.
11. `test()`.
12. Devuelve la primera aparición del patrón definido en la expresión regular.

13. Pertenece al objeto `String` y opera de modo similar al método `exec()` del objeto `RegExp`, salvo que devuelve una matriz de resultados de todas las apariciones del patrón buscado.
14. Mediante una expresión regular se define un grupo de caracteres y después el método `replace()` permite reemplazar cualquier elemento del grupo de la expresión por un carácter determinado que se pasa como segundo parámetro del método.

## Respuestas del capítulo 7

1. Nos permite saber si una variable contiene un valor numérico o no.
2. Según se quiera contemplar decimales o no hay dos funciones:

```
var miCadena= "5000.66";
var miEntero=parseInt(miCadena); // miEntero será 5000
var miFloat=parseFloat(miCadena); // miFloat será 5000.6
```

3. La función `isNaN()` analiza todo el contenido de la variable, de principio a fin, para determinar `false` o `true` si la variable es o no es un número. No devuelve nunca un valor que no sea `true` o `false`. En cambio, las funciones `parseInt()` o `parseFloat()` devuelven la parte numérica encontrada en la parte inicial de una variable, analizándola de izquierda a derecha.
4. Hay dos modos:  
Con la función `toString()`  
Simplemente concatenando al número una cadena de caracteres.
5. Básicamente todas las funciones matemáticas.
6. No, el objeto se crea automáticamente.
7. Falso. El resultado es `-Infinity` o `Infinity`.
8. Falso. Tiene también propiedades con las principales constantes matemáticas, por ejemplo, `Math.PI`.

## Respuestas del capítulo 8

1. Se utiliza la opción `fullscreen` para abrir la ventana popup.
2. El método `reload()` recarga la página en los navegadores más modernos; el siguiente es un código alternativo que funciona en todos los navegadores:

# Índice

## A

abort() 301  
abs() 123  
acceso a los caracteres de una cadena 97  
acceso a un nodo 264  
acceso a una propiedad 61  
acceso por índice a forms 198  
acos() 123  
action 199  
ActiveXObject 298  
Ajax 287, 297  
Ajax en acción 294  
alert() 140  
almacenamiento de caché 40  
alt 172  
ámbito de las variables 42  
ámbito del lenguaje 17  
análisis del modelo DOM 226  
anchor() 96  
appName 142  
appendChild(n1 ) 263  
appendData(c1) 272  
appMinorVersion 142  
appName 142  
appVersion 142  
archivo externo js 38  
archivos cookies 160  
archivos externos 40  
Array() en JavaScript 72  
arreglos 71  
asignarCookie() 163  
asin() 123  
asincrónica, programación 301  
Asynchronous JavaScript and XML, Ajax 287  
atan() 123  
Attr 260  
attributes 262

## B

back() 148  
barra de direcciones 151  
barra de estado 151

barra de links 151  
barra de menú 151  
barras de desplazamientos 151  
big() 96  
blink() 96  
bloque de instrucciones 41  
blur() 135, 203  
bold() 96  
BOM 134, 157  
booleano, tipo 43  
border 172  
break, sentencia 46  
bubbles 279  
bucles de ejecución en JavaScript 47  
búsqueda dentro de una cadena 100

## C

caché 40  
caller, propiedad 52  
cambio de posición de la ventana 149  
cambio de tamaño de la ventana 148  
cancelable 279  
case() 46  
CDATA 260  
CDATASection 260, 273  
ceil() 123  
cells 281  
ciclo petición/respuesta 289  
clase del objeto 60  
clase style 231, 316  
className 232  
cláusula case() 46  
clearInterval() 135  
clearTimeout() 135  
clientX 280  
clientY 280  
cloneNode(b1) 263  
close() 135  
closed 138  
codificación de caracteres 99  
codificación de la expresión regular 104  
codificación JavaScript 20  
código ASCII 98  
códigos de.nodeType 262  
color del texto 249  
combinación de condiciones 45  
comentarios 34  
Comment 273  
comparación de valores 45

- compatibilidad puesta en duda 19
- complementos en los navegadores 21
- complete 172
- componentes del formulario 202
- comunicación entre ventanas 152
- concat() 75
- concatenación de cadenas 94
- concatenación de matrices (concat()) 75
- conexión HTML/JavaScript mediante eventos 29
- confirm() 138, 188
- constantes matemáticas 125
- contenido de un formulario 198
- contenido HTML de un elemento 231
- control de bucles 47
- control de visibilidad por código 220
- conversión a número 120
- conversión a cadena (joint()) 76
- conversión a mayúsculas o minúsculas 100
- convierte a un número con decimales 120
- convierte a un número entero 120
- cookieEnabled 143
- cookies 160
- cookies y seguridad 160
- copia de objetos 73
- corrección de errores 32
- cos() 123
- cpuClass 142
- creación de objetos en JavaScript 64
- crear un objeto con new 65
- createAttribute() 268
- createAttributeNS() 268
- createCDATASection(d1) 268
- createComment() 268
- createDocumentFragment() 268
- createElement() 268
- createElementNS() 268
- createEntityReference() 269
- createTextNode() 269
- ctrlKey 280
- currentTarget 279

## CH

- CharacterData 272
- charAt() 95
- charCodeAt() 95, 98
- childNodes 262

## D

- Date en JavaScript 82
- defaultStatus 138
- definición de expresiones regulares 104
- deleteCell() 281
- deleteData() 272
- deleteRow() 280
- desigualdad 45
- detección del clic del ratón 189
- DHTML 225
- diferencia entre mayúsculas y minúsculas 43
- dimensionamiento 241
- display 235
- división por cero 125
- do en JavaScript 47, 307
- do/while, sentencia 50
- Document 260
- document en JavaScript 230, 316
- document.cookie 164
- documentElement 268
- DocumentFragment 260
- documento DOM 259
- DocumentType 260, 273
- DOM 226
- DOM Inspector 226, 261
- DTD 260

## E

- editores de texto 20
- efectos de la tecnología Ajax 288
- ejecución de una aplicación JavaScript 26
- ejemplo: Zona horaria internacional 88
- elegir archivos para cargar 215
- Element 260
- elementos superpuestos 246
- eliminación de cookies 168
- eliminación del primer elemento (shift()) 77
- eliminación del último elemento (pop()) 76
- EMBED 194, 315
- entype 199
- entidad en el DTD 260
- entities 273
- Entity 260, 274
- EntityReference 260
- entorno de pruebas 25
- entornos diferentes 20
- entrada de datos 192
- escape de caracteres 101
- escape() 101

escapes 93  
escritura de cookies 163  
esquema del documento (DTD) 260  
estilos de cursor 252  
estructura arbolada 259  
estructura de un documento HTML 258  
estructura del código 33  
getAttributeNode() 271  
eval() 101, 310  
evaluación aritmética 55  
evaluación de una cadena 101  
event 279  
evento onClick() 204  
eventos del objeto form 199  
eventos en el modelo DOM 278  
eventos en JavaScript 175  
exec() 107, 108  
exp() 123  
expresiones regulares 102  
eXtended Markup Language, XML 287  
extensión para desarrollador Web 22

## F

fileSize 172  
firstChild 262  
fishing 18  
fixed() 97  
floor() 123  
focus() 135, 203  
font-family 249  
font-size 249  
font-style 250  
font-weight 250  
fontcolor() 97  
fontSize() 97  
for ... in 74  
for ...in en JavaScript 62  
for, atributo de label 210  
for en JavaScript 47, 307  
for, sentencia 47  
form, objeto 198  
formateo de números 126  
forward() 148  
frames 155  
fromCharCode() 95, 98  
función de retorno (callback) 292  
función llamante 52  
funciones 50  
funciones en JavaScript 51

## G

gestión de errores en JavaScript 180  
gestión del ratón 185  
gestión del teclado 190  
get, method= 199  
getAllResponseHeaders() 301  
getAttribute() 271  
getAttributeNode() 271  
getDate() 83  
getDay() 83  
getElementById() 231  
getElementById() 269, 318  
getElementsByName() 231  
getElementsByTagName() 231, 269, 318  
getElementsByTagNameNS() 269, 318  
getFullYear() 83  
getHours() 83  
getMilliseconds() 83  
getMinutes() 83  
getMonth() 83  
getNamedItem(n1): 267  
getNamedItemNS(ns1, n1) 268  
getResponseHeader(nombreHeader) 301  
getSeconds() 83  
getTime() 83  
getTimezoneOffset() 83  
getUTC...() 83  
getYear() 83  
GMail 293  
go() 148  
Greenwich 82

## H

hackers 198  
hasChildNodes() 263  
hash 145  
height 172  
herramientas de desarrollo de Internet Explorer 23  
history 147  
home() 135  
hostname 144  
href 145  
HTML dinámico 225

## I

if en JavaScript 46  
if, sentencia 44  
if/else, sentencia 46

igualdad 45  
Image 171  
importNode() 269  
impresión de la pantalla 149  
inclusión de código JavaScript 38  
inclusión de elementos por el final (push())  
76  
incremento/decremento por unidades 49  
indefinido 43  
indexOf() 95, 100  
innerHeight 138  
innerWidth 138  
input button 204  
input checkbox 212  
input file 215  
input hidden 206  
input password 207  
input radio 208  
input submit 205  
input text 202  
inserción de elementos (unshift()) 79  
inserción de etiquetas HTML 112  
insertBefore() 265  
insertCell() 281  
insertData() 273  
insertRow() 280  
interacción con otros lenguajes 163  
internalSubset 273  
inversión del array (reverse()) 77  
isNaN() 120, 193  
italics() 97  
item() 267

## **J**

JavaScript no es Java 17  
JavaScript, un lenguaje cliente 16  
joint() 76, 309  
js archivo externo 38  
JSON 288, 318

## **L**

lastChild 262  
lastIndexOf() 96, 100  
lectura de cookies 164  
length 148, 267  
lenguaje interpretado 26  
lenguaje orientado a objetos 60  
limitaciones de Ajax 297  
limitaciones de las cookies 162

link(url) 97  
lista de valores 217  
lo que JavaScript no hace 17  
localName 262  
location 145  
location en JavaScript 144  
log() 123  
longitud de los nombres 36

## **LL**

llamada recursiva 51

## **M**

malsoftware 18  
mantenimiento evolutivo 32  
match() 108, 109  
Math 122  
matrices 71  
matrices asociativas 74  
max() 123  
mayor o igual 45  
mayor que 45  
mayúsculas 43  
mayúsculas/minúsculas 100  
menor o igual 45  
menor que 45  
mensaje informativo con respuesta 188  
meridiano de Greenwich 82  
metakey 280  
method 199  
método reset() 199  
método submit() 199  
métodos de un objeto 61  
métodos del objeto Array 75  
métodos del objeto location 147  
métodos vinculados al formateo HTML 96  
métodos y propiedades del objeto form 198  
mimeType 143  
min() 123  
minúsculas 43  
modelo Ajax de una aplicación Web 290  
modelo clásico de una aplicación Web 289  
modelo de objeto de navegador, BOM 134,  
258  
modelo de objeto documento 226  
modelo DOM 259  
modificaciones al árbol de nodos DOM 265  
motores de búsqueda 56  
moveBy() 137

moveTo() 136

## N

name 199

NamedNodeMap 260

namespaceURI 262

NaN, not a number 44, 120

navegación por el documento 276

necesidad del uso de arrays 71

new, operador 65

nextSibling 262

Node, objeto 260

nodeList 260

nodeName 262

nodeType 262

nodeValue 262

nodo 259, 260

nodo de nivel superior 259

nodo padre 260

nodo raíz 259

nodos hermanos 261

nomenclatura de variables y funciones 35

Notation 260, 274

notationName 274

numérico, tipo 43

## O

objetivo de los hackers 198

objeto Array() 72

objeto Date 82

objeto document 230, 316

objeto event 279

objeto form 198

objeto history 147

objeto Image 171

objeto location 144

objeto Math 122

objeto navigator 140

objeto screen 143

objeto String 94

objeto XMLHttpRequest 298

objetos DOM 259

objetos elements 202

objetos en JavaScript 64

objetos propios de JavaScript 60

onAbort 176

onBlur 176

onBlur() 203

onClick 176

onClick() 204

onContextMenu 176

onChange 176

onChange() 203

onDbClick 176

onDragDrop 176

onError 177

onerror 181

onFocus 177, 203

onKeyDown 177, 204

onKeyPress 177

onKeyPress() 204

onKeyUp 177

onLine 143

onLoad 177

onMouseDown 177

onMouseMove 177

onMouseOut 177

onMouseOver 177

onMouseUp 177

onReset 177, 199

onreset 201

onResize 177

onSelect 177

onSubmit 177, 199

onsubmit 201

onUnload 177

open() 299

opener 153, 312

operador ternario en JavaScript 47

operador typeof 43

operadores de comparación de valores 45

operadores lógicos Y u O 45, 307

ordenamiento del array (sort()) 78

outerHeight 138

outerWidth 138

ownerDocument 262

ownerElement 272

## P

palabra clave this 201

parámetros de la función 51

parentNode 262

parseFloat() 120

parseInt() 120

pathname 145

petición/respuesta 289

platform 142

plugins 142

pop() 76  
popup en JavaScript 150  
port 145  
posición de la ventana 149  
posicionamiento 238  
post, method= 199  
pow() 123  
precarga y visualización dinámica de imágenes 172  
previousSibling 262  
print() 135, 149  
ProcessingInstruction 260  
programación asincrónica 301  
prompt() 139, 192, 193  
protocol 145  
publicID 274  
push() 76

## Q

QName 262

## R

random() 123  
readyState 300  
recorte del array (splice()) 77  
redireccionamiento de páginas 147  
reemplazo parcial del array (splice()) 78  
referencia al elemento de raíz 275  
RegExp 103  
reload() 147  
removeAttribute(n1) 271  
removeAttributeNode(a1) 271  
removeChild() 266  
removeChild(n1) 263  
removeNamedItem(n1) 268  
removeNamedItemNS(ns1, n1) 268  
replace() 111, 147  
replaceChild(n1, n2) 264  
replaceData(d1, l1, c1) 273  
reproducción de Sonido 194  
reset() 199  
resizeBy() 136  
resizeTo() 136  
responseText 300  
reverse() 77  
robo de información personal 18  
round() 123

## S

sangría del código 33  
screen 143  
screenLeft 138  
screenTop 138  
screenX 138, 280  
screenY 138, 280  
script JavaScript 37  
script, modo indirecto 38  
scrollBy() 136  
scrollTo() 136  
search 145  
seguridad en la entrada de datos 221  
selección múltiple 219  
select, objeto 217  
select() 203  
send() 300  
sentencias JavaScript 44  
setAttribute(n1, v1) 271  
setCookie() 163  
setDate() 85  
setFullYear() 85  
setHours() 85  
setInterval() 88, 137  
setMilliseconds() 85  
setMinutes() 85  
setMonth() 85  
setNamedItem(n1) 268  
setNamedItemNS(n1) 268  
setRequestHeader(nombre, valor) 301  
setSeconds() 85  
setTime() 85  
setTimeout() 87, 135, 137  
setTimeout(tiempo) 301  
SGML 257  
shift() 77  
shiftKey 280  
sin() 123  
slice() 77, 95  
small() 97  
software indeseado 18  
sonido 194, 315  
sort() 78  
specified 272  
splice() 78  
split() 109  
splitText(d1) 273  
sqrt() 123  
src 172

Standard Generalized Markup Language 257  
status 138  
stop() 137  
strike() 97  
String en JavaScript 94  
string, tipo 43  
style 231, 316  
sub() 97  
subcadenas 99  
subir un archivo al servidor 215  
submit() 199  
substr() 95, 99  
substring() 95, 99  
substringData() 273  
sup() 97  
switch, sentencia 46  
systemID 274  
systemLanguage 143

## T

tagname{} 232  
tamaño de la ventana 148  
tan() 123  
tareas asincrónicas 290  
tareas de mantenimiento 32  
target 279  
tBodies 280  
teclado 190  
tecnología Ajax 287  
tecnologías relacionadas con Ajax 294  
test() 106  
Text 260, 273  
text-decoration 250  
textarea 216  
this, palabra clave 65  
tipo de las variables 43  
tipos de nodos DOM 260  
title 172  
toLowerCase() 96  
toString() 122  
toUpperCase() 96  
transferencia de servidor local a remoto 24  
troyanos 18  
typeof 43

## U

unescape() 102  
unshift() 79  
userAgent 143

userLanguage 143  
userProfile 142  
uso de expresiones regulares 197  
uso del caché 40

## V

validación en el servidor 197  
valores fuera de rango 197  
variables 41  
vectores 71  
ventanas emergentes 150  
verificación de una dirección de correo 114  
virus 18  
visibilidad 235  
visualización dinámica de imágenes 172

## W

W3C 295  
Web Developer 208  
while en JavaScript 47, 307  
while, sentencia 49  
width 172  
window.onerror 181  
with en JavaScript 63  
worms 18

## X

XML 287, 288, 318  
XMLHttpRequest 298  
XPath 296  
XSLT 288, 318

## Z

z-index 246

