JavaScript

Paso a paso

Edgar D'Andrea

Tabla de contenidos

Índice general

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 utiliza 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

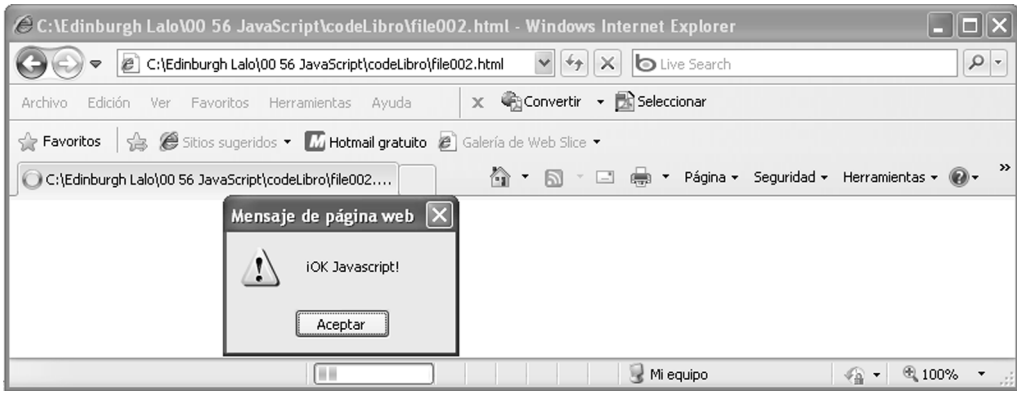
Como en todos los lenguajes de programación, el nombre que se asigne a variables y funciones tiene la habitual restricción que impide el uso de las palabras reservadas del lenguaje, básicamente, el nombre de las sentencias propias de JavaScript.

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

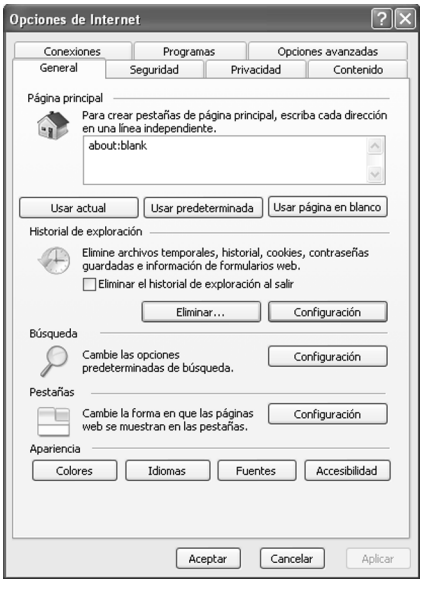
También es posible integrar código JavaScript mediante una referencia a un archivo externo de extensión js que contiene las funciones utilizadas en la página:

```
<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.

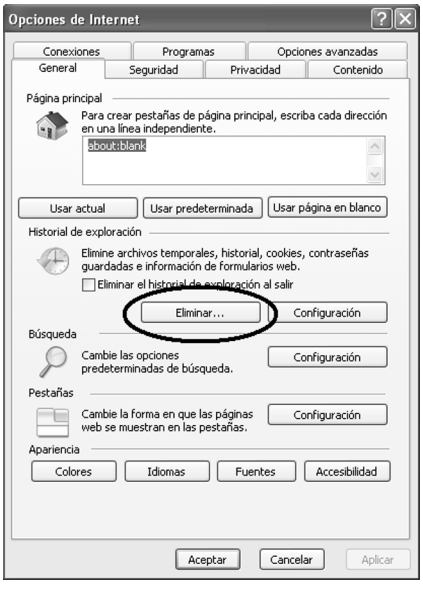
Uso de archivos externos

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 pa

ra 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:

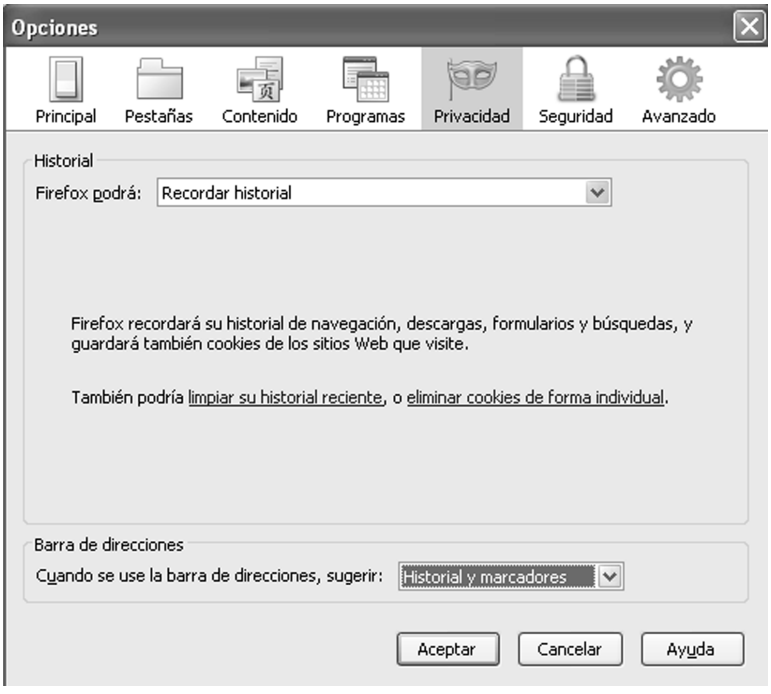
- Nombre del archivo cookie para si identificación
- Un valor asociado al nombre
- Una fecha de caducidad
- Un dominio para el que el archivo cookie tiene validez
- Una ruta que define el acceso de la página para la que el archivo cookie es legible
- Un indicador de seguridad que hace que el archivo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

Cookies y seguridad

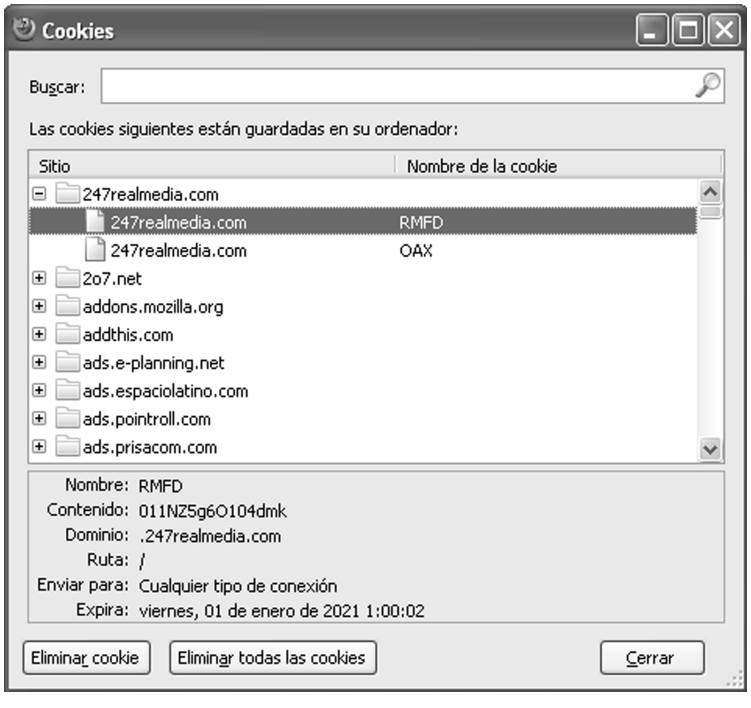
Los archivos cookies tienen una excesiva mala fama entre los usuarios lo que ha llevado a que los navegadores incluyan siempre la opción de rechazar el uso de cookies. La realidad es que la funcionalidad de los archivos cookies no debería generar tantas preocupaciones. Esto tiene su efecto en el diseño 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:



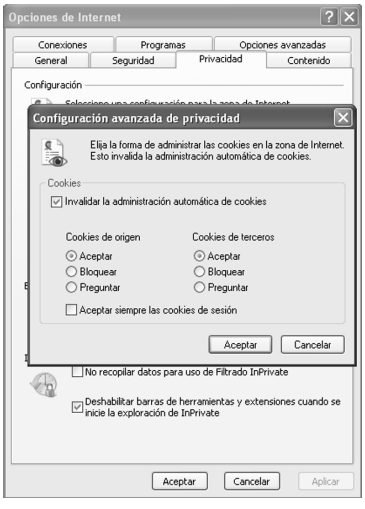
A este cuadro de diálogo se accede desde Herramientas/Opciones en la ficha Privacidad. Desde este cuadro se puede consultar el estado de los archivos cookies.



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:

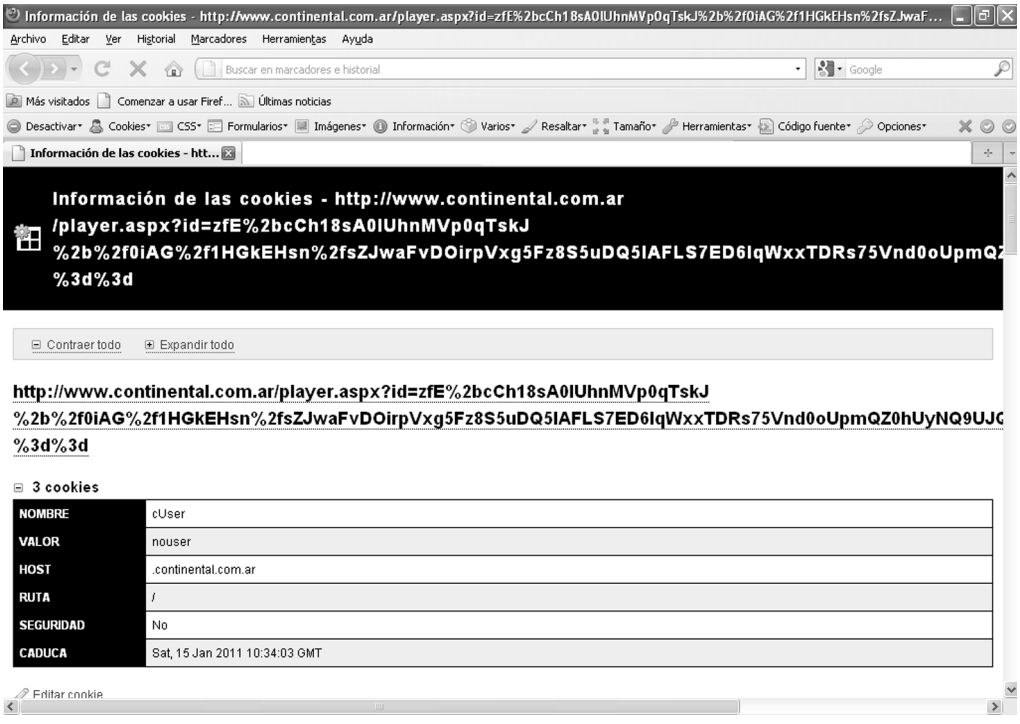
```
function leerCookie(nombre) {
// si la propiedad no tiene información se
// devuelve null
if (document.cookie.length== 0) {
     // no hay cookie registrado
     return null;
}
// separación de la cadena en grupos
// clave=valor
var sep1 = new RegExp("; ", "g");
var listaCookies = document.cookie.split(sep1);
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:



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 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 esa 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], "<BR />");
            }
     }
}

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

CH

D

E

F

G

H

I

J

L

LL

M

Q

R

S

T

U

V

W

X

Z