

3

Utilización de los objetos predefinidos de JavaScript

OBJETIVOS DEL CAPÍTULO

- ✓ Conocer cuáles son los principales objetos predefinidos de JavaScript.
- ✓ Comprender las propiedades y métodos de los principales objetos de JavaScript.
- ✓ Aprender a generar código HTML desde sentencias JavaScript.
- ✓ Dominar el uso de los marcos de HTML y aprender a realizar interacciones entre ellos con JavaScript.
- ✓ Manipular y gestionar la creación y apariencia de las ventanas del navegador, además de la comunicación entre ellas.

Hasta el momento, en los capítulos anteriores del libro, hemos realizado operaciones de programación que se han limitado a asignar valores a variables, manipular variables, calcular expresiones o usar sentencias condicionales. JavaScript fue diseñado especialmente para realizar otro tipo de actividades como, por ejemplo, manipular los navegadores y las páginas web.

Algunas de las operaciones más comunes para las cuales se diseñó JavaScript son: abrir una nueva ventana en el navegador, escribir un texto en un documento, redirigir el navegador a otra ubicación, validar los datos de un formulario o cambiar la página de un marco. La realización de algunas de estas actividades las abordaremos en capítulos posteriores de este libro. Lo importante en este punto es entender que conceptos como ventana, documento, ubicación, formulario y marco se denominan objetos predefinidos de JavaScript. Dichos objetos son elementos programables que podemos manipular para cambiar sus propiedades, realizar tareas a través de sus métodos o ejecutar un evento relacionado con el mismo objeto. Propiedades, métodos y eventos son conceptos fundamentales para comprender el funcionamiento de un objeto.

- Las propiedades son las características de un objeto. Por ejemplo, algunas de las propiedades de una página son su título (`title`) o su color de fondo (`bgColor`).
- Los métodos son funciones o tareas específicas que pueden realizar los objetos. Por ejemplo, el método `sqrt()` del objeto llamado `Math` realiza el cálculo de la raíz cuadrada de un número.
- Los eventos son situaciones que pueden llegar a realizarse o no. Por ejemplo, podemos detectar y decidir qué hacer cuando el usuario pulse el botón derecho del ratón. Cada objeto puede reconocer una serie de eventos.

Para una mejor comprensión de estos conceptos podemos pensar en un término cotidiano como es un televisor y considerarlo como un objeto:

- El color, el fabricante, el precio o el tamaño son características que se consideran propiedades del televisor.
- Con un televisor es posible ver programas, en algunos casos grabarlos, o consultar la información del teletexto. Estas tareas serían los métodos del televisor.
- Cuando pulsamos el botón de encendido o pulsamos el botón para cambiar de canal, el televisor responde de una forma predeterminada que corresponde a los eventos pertenecientes al televisor.

La suma de propiedades, métodos y eventos definen una especificación general del objeto que en este caso hemos llamado televisor.

En este capítulo estudiaremos principalmente los dos primeros conceptos de los objetos, es decir, las propiedades y los métodos. Inicialmente estudiaremos los objetos predefinidos del lenguaje, haciendo una distinción entre aquellos que están principalmente relacionados con el navegador y aquellos que están relacionados con los elementos HTML. Gracias al estudio y conocimiento de estos objetos podremos introducir temas como la generación de elementos HTML, la interacción entre los marcos de una página web y por último, la gestión de las ventanas del navegador.

3.1 OBJETOS NATIVOS DE JAVASCRIPT

JavaScript proporciona una serie de objetos definidos nativamente que no dependen del navegador en el que los utilicemos y que, por tanto, podemos manipular en cualquier momento.

La creación de un objeto se lleva a cabo a través de la palabra clave new. La sintaxis para la creación de un objeto es la siguiente:

```
var mi_objeto = new Object();
```

A la variable llamada `mi_objeto` se le asigna una nueva (new) instancia de un objeto que puede ser cualquiera de los objetos predefinidos de JavaScript o los definidos por el usuario. En el siguiente capítulo abordaremos el tema de la creación de objetos personalizados. Dentro del paréntesis introducimos los parámetros necesarios para la inicialización del objeto. En JavaScript se accede a las propiedades y a los métodos de los objetos de forma similar a los demás lenguajes de programación, es decir, con el operador punto (“.”).

```
mi_objeto.nombre_propiedad;  
mi_objeto.nombre_función([parámetros]);
```

Los objetos de JavaScript se ordenan de modo jerárquico, tal y como puede vemos en la Figura 3.1. En lo más alto de la jerarquía se encuentra el objeto `Window`, dado que representa la ventana del navegador. Este objeto posee más propiedades y métodos que los demás objetos de JavaScript. En el mismo nivel se encuentran los objetos predefinidos del lenguaje.

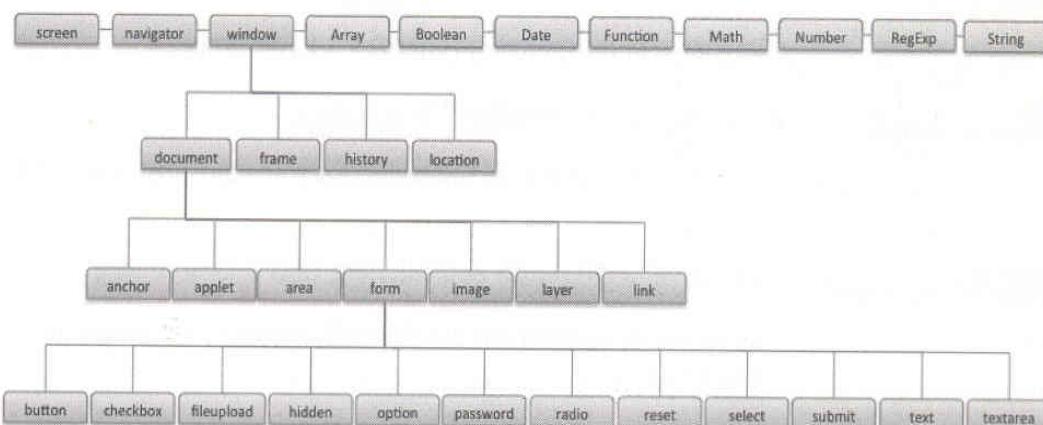


Figura 3.1. Jerarquía de los objetos de JavaScript

A continuación describimos las propiedades y los métodos de algunos de los principales objetos nativos de JavaScript, es decir, los objetos Date, Math, Number y String, dejando para el siguiente apartado los objetos que están directamente relacionados con el navegador.

3.1.1 EL OBJETO DATE

El objeto Date nos permite realizar controles relacionados con el tiempo en las aplicaciones web. Este objeto no presenta ninguna propiedad. Sin embargo, tiene una serie de métodos que podemos dividirlos en tres subconjuntos:

- **Métodos de lectura.** Estos métodos son aquellos que empiezan con el prefijo `get`. La principal función de este subconjunto de métodos es la de consultar las diferentes partes de una instancia del objeto Date.
- **Métodos de escritura.** Los métodos de escritura empiezan con el prefijo `set`. Tal y como indica su prefijo, la utilidad de estos métodos es inicializar las diferentes partes de una instancia del objeto Date.
- **Métodos de conversión.** Estos métodos convierten objetos de tipo Date en cadenas de texto o en milisegundos según los diferentes criterios que veremos a continuación.

Tabla 3.1 Métodos del objeto Date

Método	Descripción
<code>getDate()</code>	Devuelve el número correspondiente al día del mes entre 1 y 31.
<code>getDay()</code>	Devuelve el número correspondiente al día de la semana entre 0 (domingo) y 6 (sábado).
<code>getFullYear()</code>	Devuelve un número de 4 cifras correspondiente al año.
<code>getHours()</code>	Devuelve la hora entre 0 y 23.
<code>getMilliseconds()</code>	Devuelve los milisegundos entre 0 y 999.← 999
<code>getMinutes()</code>	Devuelve los minutos entre 0 y 59.
<code>getMonth()</code>	Devuelve el mes entre 0 (enero) y 11 (diciembre).
<code>getSeconds()</code>	Devuelve los segundos entre 0 y 59.
<code>getTime()</code>	Devuelve el número de milisegundos transcurridos desde el 1 de enero de 1970.
<code>getTimezoneOffset()</code>	Devuelve la diferencia entre la hora GMT y la hora local.
<code>getUTCDate()</code>	Devuelve el número del día del mes, respecto al tiempo universal coordinado.
<code>getUTCDay()</code>	Devuelve el número de la semana, respecto al tiempo universal coordinado.
<code>getUTCFullYear()</code>	Devuelve el número del año, respecto al tiempo universal coordinado.
<code>getUTCHours()</code>	Devuelva la hora, respecto al tiempo universal coordinado.
<code>getUTCMilliseconds()</code>	Devuelve los milisegundos, respecto al tiempo universal coordinado.
<code>getUTCMinutes()</code>	Devuelve los minutos, respecto al tiempo universal coordinado.
<code>getUTCMonth()</code>	Devuelve el número del mes, respecto al tiempo universal coordinado.

<code>getUTCSeconds ()</code>	Devuelve los segundos, respecto al tiempo universal coordinado.
<code>parse()</code>	Analiza una fecha y devuelve el número de milisegundos pasados desde el 1 de enero de 1970 hasta la fecha analizada.
<code> setDate ()</code>	Establece el valor del día del mes.
<code> setFullYear ()</code>	Establece el valor del año.
<code> setHours ()</code>	Establece el valor de la hora.
<code> setMilliseconds ()</code>	Establece el valor de los milisegundos.
<code> setMinutes ()</code>	Establece el valor de los minutos.
<code> setMonth ()</code>	Establece el valor del mes.
<code> setSeconds ()</code>	Establece el valor de los segundos.
<code> setTime ()</code>	Establece una fecha, agregando o sustrayendo un número específico de milisegundos hasta o desde la medianoche del 1 de enero de 1970.
<code> setUTCDate ()</code>	Establece el valor del día del mes, respecto al tiempo universal coordinado.
<code> setUTCFullYear ()</code>	Establece el valor del año, respecto al tiempo universal coordinado.
<code> setUTCHours ()</code>	Establece el valor de la hora, respecto al tiempo universal coordinado.
<code> setUTCMilliseconds ()</code>	Establece el valor de los milisegundos, respecto al tiempo universal coordinado.
<code> setUTCMilliseconds ()</code>	Establece el valor de los minutos, respecto al tiempo universal coordinado.
<code> setUTCMonth ()</code>	Establece el valor del mes, respecto al tiempo universal coordinado.
<code> setUTCSeconds ()</code>	Establece el valor de los segundos, respecto al tiempo universal coordinado.
<code> toDateString ()</code>	Convierte la fecha del objeto <code>Date</code> en una cadena de caracteres.
<code> toLocaleDateString ()</code>	Convierte la fecha del objeto <code>Date</code> en una cadena de caracteres, según las convenciones locales.
<code> toLocaleTimeString ()</code>	Devuelve la parte del tiempo de un objeto <code>Date</code> en una cadena según las convenciones locales.
<code> toLocaleString ()</code>	Convierte un objeto <code>Date</code> en una cadena de texto, según las convenciones locales.
<code> toTimeString ()</code>	Convierte la parte de tiempo de un objeto <code>Date</code> en una cadena.
<code> toUTCString ()</code>	Convierte un objeto <code>Date</code> en una cadena según el tiempo universal.
<code> UTC ()</code>	Devuelve el número de milisegundos en una cadena de tipo fecha desde la medianoche del 1 de enero de 1970.

Todas las operaciones relacionadas con fechas recaen sobre el objeto Date. Para instanciar un objeto de este tipo existen dos formas: crear un objeto con la fecha actual y crear un objeto con una fecha diferente a la actual. Esto depende de los parámetros que definamos a la hora de crear el objeto. A continuación, veremos un ejemplo con dos objetos de tipo Date en el cual realizamos una operación de resta entre ellos.

```
<script type="text/javascript">
    var fecha_actual = new Date();
    var fecha_maya = new Date(2012, 11, 21);
    alert("La fecha actual es: " + fecha_actual);
    alert("El calendario Maya termina el: " + fecha_maya);
    var tiempo_restante = fecha_maya - fecha_actual;
    alert("Quedan " + tiempo_restante + " milisegundos para
        que termine el calendario Maya");
</script>
```

3.1.2 EL OBJETO MATH

El objeto Math permite realizar operaciones matemáticas en JavaScript. En ejemplos anteriores hemos realizado operaciones matemáticas básicas como son las operaciones aritméticas. Con el objeto Math es posible realizar otro tipo de tareas más complejas como, por ejemplo, calcular raíces cuadradas o funciones trigonométricas. Las propiedades almacenan una serie de constantes que son útiles para realizar ciertos cálculos matemáticos. Estas constantes deben ser conocidas por todo aquel que haya realizado algún curso de matemáticas avanzadas.

Tabla 3.2 Propiedades del objeto Math

Propiedad	Descripción
E	Devuelve la constante de Euler.
LN2	Devuelve el logaritmo natural de 2.
LN10	Devuelve el logaritmo natural de 10.
LOG2E	Devuelve el logaritmo de E en base 2.
LOG10E	Devuelve el logaritmo de E en base 10.
PI	Devuelve el valor de Pi.
SQRT1_2	Devuelve la raíz cuadrada de 1/2.
SQRT2	Devuelve la raíz cuadrada de 2.

Tabla 3.3 Métodos del objeto Math

Método	Descripción
abs()	Devuelve el valor absoluto.
acos()	Devuelve el arcocoseno.
asin()	Devuelve el arcoseno.
atan()	Devuelve el arctangente.
ceil()	Devuelve el número entero superior.
cos()	Devuelve el coseno.
exp()	Devuelve el exponencial.
floor()	Devuelve el número entero inferior.
log()	Devuelve el logaritmo natural.
max()	Devuelve el número máximo entre los números pasados como argumento.
min()	Devuelve el número mínimo entre los números pasados como argumento.
pow()	Devuelve el resultado de un número elevado a una potencia pasada como argumento.
random()	Devuelve un número aleatorio entre 0 y 1.
round()	Redondea un número al número entero más próximo.
sin()	Devuelve el seno.
sqrt()	Devuelve la raíz cuadrada.
tan()	Devuelve la tangente.

Para utilizar el objeto Math no necesitamos definir un objeto con el constructor new. Las propiedades y los métodos son accesibles a través del nombre del tipo de objeto seguido por el nombre de la propiedad o el método. Los datos sobre los cuales queramos realizar la operación matemática, debemos pasarlos como argumentos de los métodos.

A continuación, utilizamos la propiedad Math.PI y el método Math.pow() para calcular el área de un círculo.

```
<script type="text/javascript">
  var r = prompt("Ingresa el radio de un círculo:");
  var area = Math.PI * Math.pow(r, 2);
  alert("El área del círculo es de: " + area + " cm²");
</script>
```

ej. 3.2

3.1.3 EL OBJETO NUMBER

Un objeto más para realizar tareas relacionadas con tipos de datos numéricos es el objeto `Number`. Es poco común crear objetos de tipo `Number` a través del constructor `new`, ya que por lo general se asignan directamente los valores numéricos a una variable. El objeto `Number` se utiliza principalmente para indicar el máximo y el mínimo valor posible que podemos representar con JavaScript e informar al usuario cuando sobrepase esos límites en alguna operación matemática.

Tabla 3.4 Propiedades del objeto `Number`

Propiedad	Descripción
<code>MAX_VALUE</code>	Devuelve el mayor número posible en JavaScript.
<code>MIN_VALUE</code>	Devuelve el menor número posible en JavaScript.
<code>NaN</code>	Representa el valor especial Not a Number.
<code>NEGATIVE_INFINITY</code>	Representa el infinito negativo.
<code>POSITIVE_INFINITY</code>	Representa el infinito positivo.

Tabla 3.5 Métodos del objeto `Number`

Método	Descripción
<code>toExponential()</code>	Convierte el número en una notación exponencial.
<code>toFixed()</code>	Formatea el número con la cantidad de dígitos decimales que pasemos como parámetro.
<code>toPrecision()</code>	Formatea el número con la longitud que pasemos como parámetro.

toString() → cambia de base

En el siguiente ejemplo muestra todas las propiedades del objeto `Number`, además de la construcción de una instancia y el uso de uno de sus métodos:

```
<script type="text/javascript">
  alert("Propiedad MAX_VALUE: " + Number.MAX_VALUE)
  alert("Propiedad MIN_VALUE: " + Number.MIN_VALUE)
  alert("Propiedad NaN: " + Number.NaN)
  alert("Propiedad NEGATIVE_INFINITY: " +
    Number.NEGATIVE_INFINITY)
  alert("Propiedad POSITIVE_INFINITY: " +
    Number.POSITIVE_INFINITY)
```

3.3
ej.

```
var N1 = new Number(3.141592653589793);
alert("Pi formateado: " + N1.toPrecision(3))
</script>
```

3.1.4 EL OBJETO STRING

El objeto *String* es aquel que permite manipular las cadenas de texto. Este objeto posee solamente una propiedad que indica el tamaño de la cadena. Sin embargo, presenta diferentes métodos que podemos aplicar a toda variable a la cual asignemos como valor una cadena de texto.

En capítulos anteriores del libro hemos utilizado diferentes ejemplos que usan cadenas de texto. A todas estas variables les podremos aplicar los diferentes métodos que vemos en la Tabla 3.7.

Tabla 3.6 Propiedades del objeto *String*

Propiedad	Descripción
<i>Length</i> <i>length</i>	Corresponde a la longitud de una cadena de texto.

Tabla 3.7 Métodos del objeto *String*

Método	Descripción
<i>anchor()</i>	Devuelve una cadena convertida en un ancla de HTML.
<i>big()</i>	Aumenta el tamaño de una cadena.
<i>blink()</i>	Crea el efecto de una cadena parpadeando.
<i>bold()</i>	Muestra una cadena en negrita.
<i>charAt()</i>	Permite acceder a un carácter en concreto de una cadena.
<i>charCodeAt()</i>	Devuelve un carácter en concreto en su formato Unicode.
<i>concat()</i>	Concatena dos o más cadenas y devuelve una nueva con dicha concatenación.
<i>fixed()</i>	Convierte una cadena en una cadena con fuente monoespaciada.
<i>fontcolor()</i>	Modifica el color de la fuente de una cadena.
<i>fontsize()</i>	Modifica el tamaño de la fuente de una cadena.
<i>fromCharCode()</i>	Convierte valores Unicode en caracteres.

<code>indexOf()</code>	Devuelve la posición de la primera ocurrencia del carácter pasado como parámetro.
<code>italics()</code>	Muestra una cadena en cursiva.
<code>lastIndexOf()</code>	Devuelve la posición de la última ocurrencia del carácter pasado como parámetro.
<code>link()</code>	Muestra una cadena como un hipervínculo HTML con el enlace le pasemos como parámetro.
<code>match()</code>	Busca una coincidencia en una cadena y devuelve todas las coincidencias encontradas.
<code>replace()</code>	Busca una coincidencia en una cadena y si existe, la reemplaza por otra cadena pasada como parámetro.
<code>search()</code>	Busca una coincidencia en una cadena y devuelve la posición de la coincidencia.
<code>slice()</code>	Extrae una parte de una cadena en base a los parámetros que indiquemos como índices de inicio y final.
<code>small()</code>	Disminuye el tamaño de una cadena.
<code>split()</code>	Corta una cadena en base a un separador que pasamos como parámetro.
<code>strike()</code>	Muestra una cadena tachada.
<code>sub()</code>	Muestra una cadena como subíndice.
<code>substr()</code>	Devuelve una subcadena en base a un índice y longitud pasados como parámetros.
<code>substring()</code>	Devuelve una subcadena en base a un índice de inicio y de final pasados como parámetros.
<code>sup()</code>	Muestra una cadena como superíndice.
<code>toLowerCase()</code>	Convierte una cadena en minúsculas.
<code>toUpperCase()</code>	Convierte una cadena en mayúsculas.



Los métodos del objeto `String` no respetan los estándares de la W3C (*World Wide Web Consortium*), organismo encargado de la estandarización de las tecnologías de Internet. Por este motivo, es importante prestar mucha atención al uso de estos métodos y, en muchos casos, es preferible el uso de un diseño basado en hojas de estilo en cascada (CSS).

En el siguiente ejemplo utilizamos algunos de los métodos que permiten establecer un formato sobre las cadenas de texto:

```
<script type="text/javascript">
    var texto = new String("Prueba de texto ");
    document.write("Número de letras de la cadena de
        texto: " + texto.length + "<br>");
    document.write("Cursiva: " + texto.italics() + "<br>");
    document.write("Negrita: " + texto.bold() + "<br>");
    document.write("Rojo: " + texto.fontcolor("#FF0000") +
        "<br>");
    document.write("Grande: " + texto.fontsize(20) + "<br>");
    document.write("Tachado: " + texto.strike() + "<br>");
</script>
```

3.4
Ej.

ACTIVIDADES 3.1



- En el apartado del objeto `Math` hemos utilizado un ejemplo para calcular el área de un círculo. Realice una aplicación similar pero que calcule el área de un triángulo en el cual la base y la altura las proporcione el usuario.

Recuerde que el área de un triángulo es igual a: $(\text{base} * \text{altura})/2$.

3.2 INTERACCIÓN DE LOS OBJETOS CON EL NAVEGADOR

En la sección anterior hemos presentado algunos de los principales objetos predefinidos de JavaScript. Sin embargo, existen otro tipo de objetos que permiten el control del navegador en sí mismo. Estos objetos nos permiten controlar diferentes características del navegador como, por ejemplo, qué mensaje mostramos en la barra de estado o cómo crear nuevas ventanas.

A continuación presentaremos los principales objetos que permiten manipular y gestionar algunas de las características del navegador.

3.2.1 EL OBJETO NAVIGATOR

El objeto `Navigator` permite identificar las características de la plataforma sobre la cual se está ejecutando la aplicación web escrita con JavaScript. En concreto, permite conocer datos como el tipo de navegador que se está utilizando, su versión y el sistema operativo del usuario. Este objeto se suele utilizar para obtener este tipo de información y en base al resultado, tomar una decisión sobre qué tipo de código ejecutar. Esto se debe a que no todos los navegadores se comportan del mismo modo con el mismo código. Por ello, en algunos casos, es necesario adaptar algún fragmento de código para cada navegador.

Tabla 3.8 Propiedades del objeto *Navigator* *Actualizar (ver W3C)*

Propiedad	Descripción
appCodeName	Devuelve el código del nombre del navegador.
appName	Devuelve el nombre del navegador.
appVersion	Devuelve la versión del navegador.
cookieEnable	Determina si las <i>cookies</i> están habilitadas o no.
platform	Devuelve la plataforma sobre la cual se está ejecutando el navegador.
userAgent	Devuelve una información completa sobre el agente de usuario, el cual es normalmente el navegador.

Tabla 3.9 Métodos del objeto *Navigator*

Método	Descripción
javaEnabled()	Informa si el navegador está habilitado para soportar la ejecución de programas escritos en Java.

En el siguiente ejemplo realizamos algunas consultas sobre el navegador, además de verificar si está preparado para la ejecución de *applets* de Java. En este caso simplemente imprimimos el resultado en la pantalla, pero en aplicaciones web avanzadas es posible realizar tareas más complejas en base a este resultado.

```

<script type="text/javascript">
    document.write("Navegador: " + navigator.appName +
    "<br>");
    document.write("Versi\xf3n: " + navigator.appVersion +
    "<br>");

    if (navigator.javaEnabled()){
        document.write("El navegador S\xcd est\xel preparado
        para soportar los Applets de Java")
    }
    else{
        document.write("El navegador NO est\xel preparado para
        soportar los Applets de Java")
    }
</script>
```

3.6
3.3

3.2.2 EL OBJETO SCREEN

El objeto Screen corresponde a la pantalla utilizada por el usuario. Este objeto cuenta con seis propiedades, aunque no posee ningún método. Todas sus propiedades son solamente de lectura, lo que significa que podemos consultar los valores de las propiedades del objeto, pero no las podemos modificar.

Tabla 3.10 Propiedades del objeto Screen

Propiedad	Descripción
availHeight	Corresponde a la altura disponible de la pantalla para el uso de ventanas.
availWidth	Corresponde a la anchura disponible de la pantalla para el uso de ventanas.
colorDepth	Corresponde al número de colores que puede representar la pantalla.
height	Corresponde a la altura total de la pantalla.
pixelDepth	Corresponde a la resolución de la pantalla expresada en bits por píxel.
width	Corresponde a la anchura total de la pantalla.

La altura y anchura disponibles para las ventanas es menor que la altura y anchura total de la pantalla, debido a que cada sistema operativo ocupa una parte de la pantalla con barras de tareas o menús.

El uso del objeto Screen y sus propiedades, es muy común en los diseñadores de páginas web que necesitan saber la resolución de la pantalla del usuario para poder adaptar sus diseños antes de cargarlos. Otro uso bastante común de este objeto, es consultar todas sus propiedades con el fin de adaptar la posición y tamaño de las ventanas emergentes que abra la aplicación web.

```

<script type="text/javascript">
    document.write("<br>Altura total: " + screen.height);
    document.write("<br>Altura disponible: " +
        screen.availHeight);
    document.write("<br>Anchura total: " + screen.width);
    document.write("<br>Anchura disponible: " +
        screen.availWidth);
    document.write("<br>Profundidad de color: " +
        screen.colorDepth);
</script>
```

3.6
EJ

3.2.3 EL OBJETO WINDOW

Como hemos mencionado anteriormente, el objeto Window se considera el objeto más importante del lenguaje JavaScript. A partir de él, podemos gestionar las ventanas del navegador y utilizar una serie de propiedades y métodos que presentamos en las siguientes tablas.

El objeto Window se considera como un objeto implícito, ya que no es necesario nombrarlo para acceder a los objetos que se encuentran en el nivel ubicado bajo su nivel de jerarquía. Por ejemplo, tal y como hemos realizado en alguna de las actividades propuestas en las secciones anteriores, si quisiésemos acceder a la propiedad que cambia el color de fondo de una página, deberíamos escribir `window.document.bgColor`. En los ejemplos anteriores hemos podido omitir la escritura del objeto Window, dado que JavaScript reconoce que todos los demás objetos están debajo de su nivel de jerarquía.

Tabla 3.11 Propiedades del objeto *Window*

Propiedad	Descripción
<code>closed</code>	Corresponde al valor booleano que indica si la ventana está cerrada o no.
<code>defaultStatus</code>	Corresponde a la cadena de texto de la barra de estado del navegador.
<code>document</code>	Corresponde al documento actual de la ventana.
<code>frames</code>	Corresponde al conjunto de marcos de la ventana.
<code>history</code>	Corresponde al conjunto de elementos que representan las URL visitadas.
<code>innerHeight</code>	Corresponde a la altura utilizable de la ventana.
<code>innerWidth</code>	Corresponde al ancho utilizable de la ventana.
<code>length</code>	Corresponde al número de <code>frames</code> de la ventana.
<code>location</code>	Corresponde a la URL de la barra de direcciones.
<code>locationbar</code>	Corresponde a la barra de direcciones del navegador.
<code>menubar</code>	Corresponde a la barra de menú del navegador.
<code>name</code>	Corresponde al nombre de la ventana.
<code>opener</code>	Corresponde a la referencia al objeto <code>Window</code> que haya abierto una ventana nueva.
<code>outerHeight</code>	Corresponde a la altura exterior de la página.
<code>outerWidth</code>	Corresponde al ancho exterior de la página.
<code>pageXOffset</code>	Corresponde a la posición horizontal de la ventana.
<code>pageYOffset</code>	Corresponde a la posición vertical de la ventana.
<code>parent</code>	Corresponde a la referencia al objeto <code>Window</code> que contiene los marcos de una página de marcos.
<code>personalbar</code>	Corresponde a la barra de herramientas personal.
<code>scrollbars</code>	Corresponde a las barras de desplazamiento vertical y horizontal.
<code>self</code>	Corresponde a la ventana actual.
<code>status</code>	Corresponde a la cadena con el mensaje que contiene la barra de estado.
<code>toolbar</code>	Corresponde a la barra de herramientas del navegador.
<code>top</code>	Corresponde a la ventana de nivel superior.

Todas estas propiedades podemos manipularlas con el fin de mostrar o no una parte de la ventana del navegador o modificar algunas características como su tamaño o posición. A lo largo del capítulo mostraremos ejemplos de cómo realizar estas operaciones. Algunos de los métodos del objeto Window ya los hemos usado en ejemplos o actividades de secciones anteriores. La mayor parte de estos métodos están relacionados con la gestión de las ventanas y la navegación de las páginas web.

Tabla 3.12 Métodos del objeto *Window*

Método	Descripción
alert()	Genera un cuadro de diálogo con un mensaje y un botón <i>Aceptar</i> .
back()	Regresa a la página anterior según el historial.
blur()	Desactiva una página.
close()	Cierra una página.
confirm()	Genera un cuadro de diálogo con los botones <i>Aceptar</i> y <i>Cancelar</i> .
find()	Realiza una búsqueda de texto en una página.
focus()	Activa una ventana.
forward()	Avanza una página según el historial.
home()	Carga la página definida como página por defecto del navegador.
moveTo()	Mueve la ventana activa.
open()	Abre una nueva ventana.
print()	Imprime una página.
prompt()	Genera un cuadro de diálogo con un cuadro de texto para que el usuario introduzca valores.
resizeTo()	Modifica el tamaño de una ventana.
setInterval()	Evaluá una expresión después de un tiempo especificado.
setTimeout()	Inicia un registro de tiempo.
scrollBy()	Mueve el contenido de una ventana en función de una cantidad especificada en píxeles.
scrollTo()	Mueve el contenido de una ventana especificando una nueva posición de la esquina superior izquierda.
stop()	Detiene una página.

Algunos de los métodos ya los hemos utilizado en ejemplos anteriores del libro y otros más completos los presentaremos en los siguientes apartados dedicados a la gestión de las ventanas.

3.2.4 EL OBJETO DOCUMENT

El objeto Document se refiere a los documentos que se cargan en la ventana del navegador. Dado que con este objeto es posible manipular las propiedades y el contenido de los elementos principales de las páginas web, probablemente sea el objeto más utilizado en los programas escritos en JavaScript.

El objeto Document cuenta con una serie de subobjetos como son los vínculos, puntos de anclaje, imágenes o formularios. Esta colección de subobjetos representa el verdadero potencial del objeto Document. A continuación presentamos las principales propiedades y métodos de dicho objeto.

Tabla 3.13 Propiedades del objeto *Document*

Propiedad	Descripción
alinkColor	Corresponde al color de los vínculos activos de la página.
anchors	Corresponde a los puntos de anclaje (etiquetas <code><a name></code>) del documento.
applets	Corresponde a los <i>applets</i> (etiquetas <code><applet></code>) Java del documento.
bgColor	Corresponde al color de fondo del documento.
cookie	Corresponde a un fichero guardado en el equipo del cliente del navegador con información sobre el usuario.
domain	Corresponde al nombre del dominio por defecto para el documento.
embeds	Corresponde a los objetos embebidos (etiqueta <code><embed></code>) en el documento.
fgColor	Corresponde al color del texto del documento.
forms	Corresponde a los formularios (etiqueta <code><form></code>) del documento.
images	Corresponde a las imágenes (etiqueta <code><images></code>) del documento.
lastModified	Corresponde a la última fecha en la que se modificó el documento.
layers	Corresponde a las capas (etiqueta <code><layer></code>) del documento.
linkColor	Corresponde al color de los enlaces aún no visitados.
links	Corresponde a los vínculos (etiqueta <code><a href></code>) del documento.
plugins	Corresponde a las referencias y llamadas de los plugins del documento.
referrer	Corresponde a la dirección del documento usado para ir al documento actual.
title	Corresponde al título (etiqueta <code><title></code>) del documento.
URL	Corresponde a la dirección del documento.
vlinkColor	Corresponde al color de los enlaces visitados.

La mayor parte de estas propiedades contiene una lista con cada uno de los elementos presentes en el documento HTML. Una vez que se obtiene la lista, es posible acceder a cada elemento para gestionar o modificar cada uno de ellos. El objeto `Document` tiene diferentes métodos bastante útiles, que resumimos en la Tabla 3.14.

Tabla 3.14 Métodos del objeto `Document`

Método	Descripción
<code>captureEvents()</code>	Intercepta un evento para que pueda ser manipulado por el documento.
<code>close()</code>	Cierra el documento activo.
<code>getSelection()</code>	Devuelve el texto seleccionado en el documento.
<code>handleEvent()</code>	Activa el manejador del evento especificado.
<code>home()</code>	Carga la página de inicio.
<code>open()</code>	Activa el documento.
<code>releaseEvents()</code>	Libera los eventos que han sido interceptados.
<code>routeEvents()</code>	Intercepta un evento y lo pasa a lo largo de la jerarquía del objeto que lo lanzó.
<code>write()</code>	Escribe datos en el documento.
<code>writeln()</code>	Escribe datos además de un salto de línea en el documento.

3.2.5 EL OBJETO HISTORY

El objeto `History` almacena las referencias de todos los sitios web visitados. Estas referencias se guardan en una lista y sus propiedades y métodos se utilizan principalmente para que el usuario de una aplicación web pueda desplazarse para adelante y para atrás. Sin embargo, al ser una lista de referencias, no podemos acceder a los nombres de las direcciones URL visitadas, ya que son información privada del usuario.

Tabla 3.15 Propiedades del objeto `History`

Propiedad	Descripción
<code>current</code>	Corresponde a la cadena que contiene la URL de la entrada actual del historial.
<code>length</code>	Corresponde al número de páginas que han sido visitadas.
<code>next</code>	Corresponde a la cadena que contiene la siguiente entrada del historial.
<code>previous</code>	Corresponde a la cadena que contiene la anterior entrada del historial.

Tabla 3.16 Métodos del objeto *History*

Método	Descripción
back()	Carga la URL del documento anterior del historial.
forward()	Carga la URL del documento siguiente del historial.
go()	Carga la URL del documento especificado por el índice que pasamos como parámetro dentro del historial.

Por ejemplo, en el fichero HTML que queramos usar este objeto, podríamos crear dos botones para que el usuario pueda desplazarse adelante o atrás según su historial de navegación.

```
<form>
  <input type="button" value="Atras"
    onClick="history.back();">
  <input type="button" value="Adelante"
    onClick="history.forward;">
</form>
```

3.7
ej

3.2.6 EL OBJETO LOCATION

El objeto Location corresponde a la URL de la página web en uso. Sus principales funciones son las de consultar las diferentes partes que forman una URL, como por ejemplo el dominio, el protocolo o el puerto. De este modo, podemos extraer cada componente de la URL y trabajar con ellos de forma separada. Gracias a este objeto podemos recargar una página, cargar una nueva o remplazar una por otra.

Tabla 3.17 Propiedades del objeto *Location*

Propiedad	Descripción
hash	Corresponde a la cadena que representa el anclaje de la URL. Es decir, la parte de la URL que va después de la etiqueta #.
host	Corresponde a la cadena que representa el nombre del dominio del servidor y el número del puerto dentro de la URL.
hostname	Corresponde a la cadena que representa el nombre del dominio del servidor dentro de la URL.
href	Corresponde a la URL completa.
pathname	Corresponde a la cadena que sigue al nombre del servidor.
port	Corresponde al número de puerto de la URL.
protocol	Corresponde al protocolo utilizado por la página web.
search	Corresponde a la cadena de búsqueda que se encuentra después de un signo de interrogación de la URL.

Tabla 3.18 Métodos del objeto *Location*

Método	Descripción
assign()	Carga un nuevo documento.
reload()	Carga de nuevo el documento actual.
replace()	Sustituye la URL del documento actual por otra URL.

En el siguiente ejemplo utilizamos tres de las propiedades del objeto *Location*, además de crear un botón que al presionarlo, llama al método *location.reload()*, con lo cual cargamos de nuevo el documento actual. Este botón podría ser una alternativa al botón *refresh* de un navegador:

```
<script type="text/javascript">
    document.write("URL completa: " + location.href +
    "<br>");
    document.write("Pathname: " + location.pathname +
    "<br>");
    document.write("Protocolo: " + location.protocol +
    "<br>");
</script>
<form>
    <input type="button" value="Recargar"
        onclick="location.reload();">
</form>
```

3.8
2j



En el ejemplo anterior hemos utilizado un controlador de eventos llamado *onclick*. Este tipo de controladores son un enlace entre los objetos del documento y el código JavaScript. La descripción y análisis de los eventos los estudiaremos más adelante.

ACTIVIDADES 3.2



- En un documento HTML cree un botón que al presionarlo, utilice el método *resizeTo(500, 500)* para modificar el tamaño de la ventana.

3.3 GENERACIÓN DE ELEMENTOS HTML DESDE CÓDIGO JAVASCRIPT

Uno de los principales objetivos de JavaScript en el desarrollo web en la parte del cliente es convertir un documento HTML estático en una aplicación web dinámica. Por ejemplo, es muy común que los *scripts* detecten el tipo y la versión del navegador que estamos utilizando y, en base a esto, escribir las etiquetas adecuadas para cada navegador. De igual modo, se suelen utilizar los valores de determinadas variables para ejecutar instrucciones que crean nuevas ventanas con contenido propio, en lugar de mostrarlo en la ventana actual. Cada ventana de un navegador presenta un documento HTML y es representada por un objeto Window que contiene un subobjeto Document. El objeto Document contiene a su vez una serie de objetos que representan todo el contenido del documento HTML, como por ejemplo el texto, las imágenes, los enlaces, los formularios, las tablas, etc.

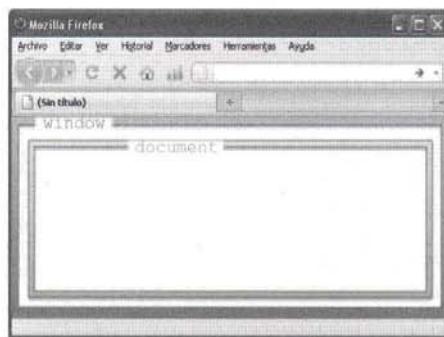


Figura 3.2. Objetos Window y Document representados en un navegador

Con JavaScript es posible manipular y acceder a los objetos que representan el contenido de una página. De este modo, en lugar de crear solamente documentos estáticos, es posible crear documentos dinámicos. Este proceso se puede realizar gracias al método `document.write()`.

En ejemplos anteriores, hemos visto que con el método `document.write()` podemos escribir un resultado por pantalla. Existen dos formas de utilizar este método para generar contenido dinámico:

- Podemos utilizarlo dentro de una secuencia de instrucciones JavaScript para mostrar un resultado en el documento de la ventana actual del navegador. Por ejemplo, es posible definir el título de una página web basándose en el nombre del sistema operativo que se utilice para abrir el documento:

```
<script type="text/javascript">
    var SO = navigator.platform;
    document.write("<h1>Documento abierto con: " + SO +
    "</h1>");
</script>
```

- La segunda forma es muy similar a la anterior. El método `document.write()` podemos utilizarlo para crear documentos de nuevas ventanas del navegador. Esto es una práctica muy común en las páginas web que utilizan ventanas emergentes. Los detalles de la creación y la comunicación entre ventanas los abordaremos en otro apartado de este capítulo. Por el momento, en el siguiente código podemos observar cómo creamos una

nueva ventana sin ningún contenido, posteriormente accedemos a su respectivo documento y, finalmente, en la ventana nueva escribimos el título de la página web según el texto que ha ingresado el usuario.

```
<script type="text/javascript">
    var texto = prompt("Ingresa un título para la nueva
        ventana: ");
    var ventanaNueva = window.open();
    ventanaNueva.document.write("<h1>" + texto + "</h1>");
</script>
```

ej. 3.10

- La generación de código HTML a partir de código JavaScript no se limita solo a la creación de texto tal y como hemos visto en los ejemplos anteriores. Podemos crear y manipular todo tipo de objetos. El siguiente ejemplo muestra cómo generar un formulario para modificar la propiedad del color de fondo de la página:

```
<script type="text/javascript">
    document.write("<form name=\"cambiacolor\">");
    document.write("<b>Selecciona un color para el fondo de
        página:</b><br>");
    document.write("<select name=\"color\">");
    document.write("<option value=\"red\">Rojo</option>");
    document.write("<option value=\"blue\">Azul</option>");
    document.write("<option
        value=\"yellow\">Amarillo</option>");
    document.write("<option value=\"green\">Verde</option>");
    document.write("</select>");
    document.write("<input type=\"button\""
        value="Modifica el color" onclick="document.bgColor
        =document.cambiacolor.color.value">");
    document.write("</form>");
</script>
```

ej. 3.11

En la Figura 3.3 vemos el resultado de la generación de una página web dinámica. En esta página el usuario puede modificar la propiedad `document.bgColor` a través de un formulario HTML que se ha creado a partir de código JavaScript.

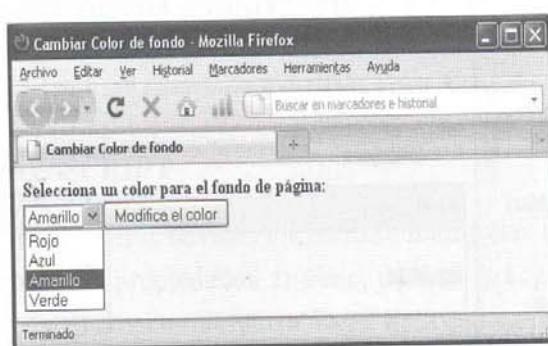


Figura 3.3. Generación de código HTML con JavaScript. Cambio de color

ACTIVIDADES 3.3



- Abra el fichero 3.11-GeneraciónCódigo3.html y modifique el ejemplo anterior agregando dos colores más al conjunto de posibles colores del fondo de la página.

3.4 APLICACIONES PRÁCTICAS DE LOS MARCOS

La mayor parte de las aplicaciones web se crean para ser ejecutadas en una sola ventana, aunque existe la posibilidad de dividir la ventana en dos o más partes independientes. En estos sectores podemos utilizar JavaScript para generar una interacción entre ellos. Estos sectores se denominan marcos. Algunas páginas web presentan una estructura en la cual una parte de la página permanece fija mientras que otra parte va cambiando en base a la navegación que se efectúa en otro marco. En realidad este efecto se produce creando diferentes páginas web y posicionando cada una de ellas en un marco diferente.

La Figura 3.4 muestra una página web que utiliza tres marcos. Esta es una página bastante conocida para aquellos que hayan realizado algún curso de programación en Java, ya que ésta es la página con la especificación API de este lenguaje. En el marco superior izquierdo se definen los paquetes de Java. Debajo de este último marco encontramos las clases, interfaces y/o excepciones pertenecientes a cada paquete y, por último, en el marco principal de la derecha, encontramos la descripción detallada de cada concepto del lenguaje. Estos tres marcos interactúan entre ellos. Por ejemplo, al elegir un paquete en el primer marco, el segundo marco cambia automáticamente para mostrar todas las clases relacionadas con dicho paquete. Si seleccionamos una de estas clases, el marco principal mostrará la descripción de dicha clase, con sus métodos, constructores, etc.

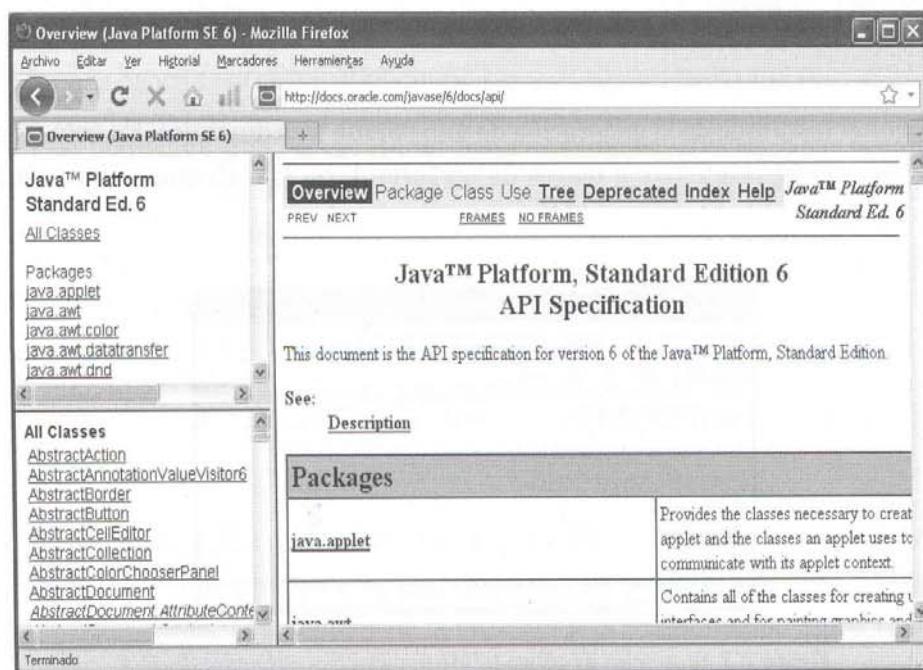


Figura 3.4. Página web definida con marcos HTML

Los marcos se definen utilizando HTML. Estos presentan la ventaja de poder crear páginas en las que es posible mantener constantemente algunos elementos como botones, enlaces, imágenes, etc. Con JavaScript podemos manipular los marcos y realizar una interacción entre ellos. De este modo es posible utilizar todas las ventajas del lenguaje sobre los marcos.

Para utilizar correctamente los marcos de HTML, necesitamos conocer un par de etiquetas con sus respectivos atributos:

- <frameset>: las funciones básicas de esta etiqueta son indicar al navegador que la página contiene marcos, definir el número de marcos que se utilizarán en la página y establecer el tamaño de cada marco. Los principales atributos son cols y rows, los cuales dividen respectivamente el conjunto de marcos vertical u horizontalmente.
- <frame>: esta etiqueta define las características de cada marco. En la Tabla 3.19 vemos sus principales atributos.

Tabla 3.19 Atributos de la etiqueta <frame>

Atributo	Descripción
frameborder	Define si mostrar o no el borde del marco.
marginheight	Permite cambiar los márgenes verticales del marco.
marginwidth	Permite cambiar los márgenes horizontales del marco.
name	Asigna un nombre al marco.
noresize	Evita que el usuario pueda modificar el tamaño del marco.
scrolling	Permite elegir si posiciona o no una barra de desplazamiento en el marco.
src	Indica la URL del documento HTML que contendrá el marco.

Todas las páginas que se dividan en al menos dos marcos, necesitarán al menos tres páginas web. La primera página web contendrá el primer marco que se denomina conjunto de marcos. Este conjunto es el que creamos utilizando la etiqueta <frameset> y contendrá las otras páginas web que se mostrarán en cada marco definido. Cada uno de los marcos definidos los creamos con la etiqueta <frame> y se considera un hijo del conjunto de marcos.

3.4.1 USO DE MARCOS CON JAVASCRIPT

Tal y como hemos mencionado anteriormente, JavaScript permite manipular los marcos. Cualquier marco puede hacer referencia a otro marco utilizando las propiedades frames, parent y top del objeto Window. Cada ventana tiene una propiedad llamada frames, la cual hace referencia a una lista que contiene todos los marcos.

El código JavaScript que ejecutamos en un conjunto de marcos puede hacer referencia a su primer marco a través del objeto frames[0], a su segundo marco con frames[1], y así sucesivamente.

Todos los marcos poseen una propiedad llamada `parent`, que hace referencia al objeto que contiene dichos marcos. Por lo tanto, es posible hacer referencia a un marco desde otro. Para ello, empleamos la propiedad `parent` seguida del nombre del marco al cual queremos acceder y, por último, el nombre del elemento que queramos manipular. Por ejemplo, debemos escribir el siguiente código si desde un marco queremos modificar el color de fondo de otro marco llamado `MarcoDePrueba`:

```
parent.MarcoDePrueba.document.bgColor
```

A continuación presentamos un ejemplo de un documento HTML en el cual hemos definido dos marcos divididos por columnas. El atributo `cols` lo hemos utilizado para que cada marco ocupe el 50% de la ventana:

```
<html>
  <head>
    <title>Ejemplos de control de marcos</title>
  </head>
  <frameset cols="50%,50%">
    <frame src="3.12a-Marcos1.html" name="Marco1" noresize>
    <frame src="3.12b-Marcos2.html" name="Marco2" noresize>
  </frameset>
  <body></body>
</html>
```

El primer marco (`Marco1`) contiene la página `3.12a-Marcos1.html`. En esta página hemos definido dos campos de selección con la etiqueta `<select>`, con el fin de elegir un color y uno de los dos marcos presentes.

```
<html>
  <body>
    <form name="form1">
      <select name="color">
        <option value="green">Verde
        <option value="blue">Azul
      </select>
      <br><br>
      <select name="marcos">
        <option value="0">Izquierda
        <option value="1">Derecha
      </select>
    </form>
  </body>
</html>
```

El segundo marco contiene la página `3.12b-Marcos2.html`. En esta página hemos agregado un botón con el controlador de eventos `onclick`, en el que hemos escrito un guión JavaScript que permite acceder a cada marco y modificar su color de fondo en base a las elecciones del primer marco.

```

<html>
  <body>
    <form>
      <input type="Button" value="Cambiar Color" onclick="
        campoColor = parent.Marco1.document.form1.color
        if(campoColor.selectedIndex==0) {
          colorin = 'green';
        }else{
          colorin = 'blue';
        }
        campoFrame = parent.Marco1.document.form1.marcos
        if(campoFrame.selectedIndex==0) {
          window.parent.Marco1.document.bgColor = colorin
        }else{
          window.parent.Marco2.document.bgColor = colorin
        }">
    </form>
  </body>
</html>

```

ej. 3.12 b



En el ejemplo anterior hemos utilizado una sentencia condicional para verificar el color elegido y el marco sobre el cual se aplicará dicho color. Existe una forma mucho más rápida y limpia de realizar este proceso gracias al uso de arrays y llamadas a funciones. Estos conceptos los veremos en el próximo capítulo del libro.

Podemos ver el resultado de la creación de estos marcos en la Figura 3.5.

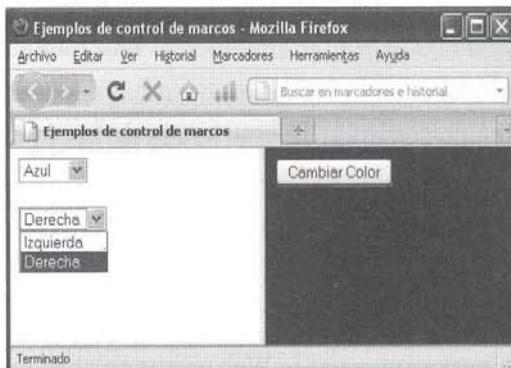


Figura 3.5. Interacción entre marcos

Una opción interesante es la de ocultar el borde de los marcos y crear páginas en las que no se note esa separación. El resultado aparece como una página web única, aunque en realidad esté dividida por varias páginas. El borde lo podemos ocultar definiendo los atributos `frameborder` y `border` iguales a cero.



El desarrollo de páginas web con múltiples marcos era una práctica común en la primera época de desarrollo con JavaScript. Actualmente, la mayoría de desarrolladores no utiliza el desarrollo de ventanas interactivas, aunque sí que es común el desarrollo con marcos en línea los cuales se denominan *iframes*.

ACTIVIDADES 3.4



- ▶ Modifique el código del ejemplo de marcos utilizado anteriormente, con el fin de ocultar el borde y que no se note la separación entre ellos.

3.5 GESTIÓN DE LAS VENTANAS

Hasta el momento, en los capítulos anteriores hemos visto cómo utilizar JavaScript para gestionar principalmente el contenido de las páginas web. En esta sección estudiaremos cómo gestionar los diferentes aspectos de las ventanas del navegador utilizando JavaScript.

Existen muchas operaciones comunes en las páginas web que visitamos a diario. Por ejemplo, es común que se abran ventanas nuevas al presionar un botón. Cada una de estas ventanas tiene tamaños, posiciones y estilos diferentes. Muchas de estas ventanas emergentes no contienen simplemente una página web estática. Su contenido suele ser dinámico y depende de las acciones realizadas en la ventana que la haya creado. En los siguientes apartados veremos cómo abrir y cerrar nuevas ventanas, cómo controlar la apariencia de las mismas y cómo crear una comunicación e interacción entre dos o más ventanas.

3.5.1 ABRIR Y CERRAR NUEVAS VENTANAS

Algo muy común que ocurre cuando navegamos por la Web es que se abra una ventana nueva cuando pulsamos un botón. Algunos sitios abren ventanas incluso sin que el usuario haga algo. Las ventanas se abren automáticamente cuando se carga una nueva página o simplemente cuando cerramos otra ventana. Tal y como hemos visto anteriormente en la descripción de los objetos de JavaScript, la ventana del navegador es un objeto más que presenta una serie de propiedades y métodos.

Las ventanas secundarias las podemos abrir utilizando solo código HTML mediante el atributo target de la etiqueta href:

```
<a href="enlace.html" target="_blank">
```

La desventaja de abrir una nueva ventana utilizando solo código HTML es que no tenemos ningún control sobre ella. La ventana se abrirá en base a la configuración predefinida del navegador del usuario. Por el contrario, con JavaScript tenemos un mayor control sobre las nuevas ventanas. Una nueva ventana vacía se crea con el método open(), el cual devuelve una referencia a dicha ventana:

```
nuevaVentana = window.open()
```

De este modo la variable llamada nuevaVentana contendrá una referencia a la ventana que hemos creado. Esta variable la podemos utilizar posteriormente para cualquier manipulación que queramos hacer sobre la ventana mientras se ejecuta JavaScript.

Las ventanas poseen diversas propiedades y métodos tal y como hemos visto en las secciones anteriores. En concreto, el método open() cuenta con los siguientes tres parámetros:

- ✓ El primero es la URL de la página web que estará contenida en la nueva ventana.
- ✓ El segundo parámetro es el nombre asignado a la nueva ventana.
- ✓ El tercer parámetro es una colección de atributos que definen la apariencia de la ventana.

De este modo, el método open() lo podemos utilizar definiendo estos tres parámetros, tal y como vemos en el siguiente ejemplo:

```
nuevaVentana = window.open("http://www.misitioWeb.com/ads",
    "Publicidad", "height=100, width=100");
```

Utilizando los anteriores parámetros, abrimos una ventana que contiene la URL y el nombre especificados en los dos primeros parámetros y tiene una altura y anchura de 100 píxeles.

A continuación presentamos un ejemplo completo de una página web en la que creamos un botón que abre una nueva ventana al hacer clic sobre él. En esta nueva ventana establecemos los atributos de altura y anchura, además de crear etiquetas HTML para generar el título de la ventana y un texto en el que especificamos las propiedades modificadas.

```
<html>
  <head></head>
  <body>
    <center><h1> Ejemplo de Apariencia de una Ventana</h1>
    <br>
    <input type="Button" value="Abre una Ventana" onclick="
      myWindow1 = window.open('', 'Nueva Ventana', 'width=300,
      height=200');
      myWindow1.document.write('<html>');
      myWindow1.document.write('<head>');
```

3.13
Ej.

```

myWindow1.document.write('<title>Ventana de Prueba
</title>');
myWindow1.document.write('</head>');
myWindow1.document.write('<body>');
myWindow1.document.writeln('Se han utilizado las
propiedades: ');
myWindow1.document.write('<li>height=200</li>
<li>width=300</li>');
myWindow1.document.write('</body>');
myWindow1.document.write('</html>');
"/>
</center>
</body>
</html>

```

Cualquiera de las ventanas que abramos, podemos igualmente cerrarlas al invocar el método `close()`. Como hemos dicho anteriormente, el método `open()` devuelve una referencia a una nueva ventana. Para cerrar la ventana debemos invocar el método `close()` sobre dicha referencia.

Para probar su funcionamiento, podemos utilizar el código del ejemplo anterior y agregar justo antes de la creación de la etiqueta `</body>` la línea:

```
myWindow1.document.write('<input type=button value=Cerrar
onClick=window.close()>');
```

La apertura de múltiples ventanas a la vez es una característica por lo general bastante molesta para los usuarios. Sin embargo, en algunos casos puede ser útil saber cómo realizar esta operación. Para ello utilizamos un bucle `for` con el fin de ejecutar el método `window.open()` todas las veces deseadas. En el siguiente ejemplo abrimos cinco ventanas a la vez que presentan a su vez un botón para poder cerrarlas:

```

<html>
<head></head>
<body>
<center><h1>Apertura de m&uacute;ltiples ventanas</h1>
<input type="Button" value="Abre m&uacute;ltiples
ventanas..." onclick="
for(i=0;i<5;i++){
    myWindow=window.open('', '', 'width=200,height=200');
    myWindow.document.write('<html>');
    myWindow.document.write('<head>');
    myWindow.document.write('<title>Ventana: '+i+
    '</title>');
    myWindow.document.write('</head>');
}

```

ej. 3. Bb

```

myWindow.document.write('<body>');
myWindow.document.write('Ventana ' + i);
myWindow.document.write('<input type=button');
    value=Cerrar onClick=window.close()>');
myWindow.document.write('</body>');
myWindow.document.write('</html>');
}
"/>
</center>
</body>
</html>

```

En la Figura 3.6 podemos ver el resultado de la ejecución del ejemplo anterior, en el cual al presionar un botón en la ventana principal, generamos cinco ventanas emergentes, cada una de ellas con un botón para cerrar cada ventana.

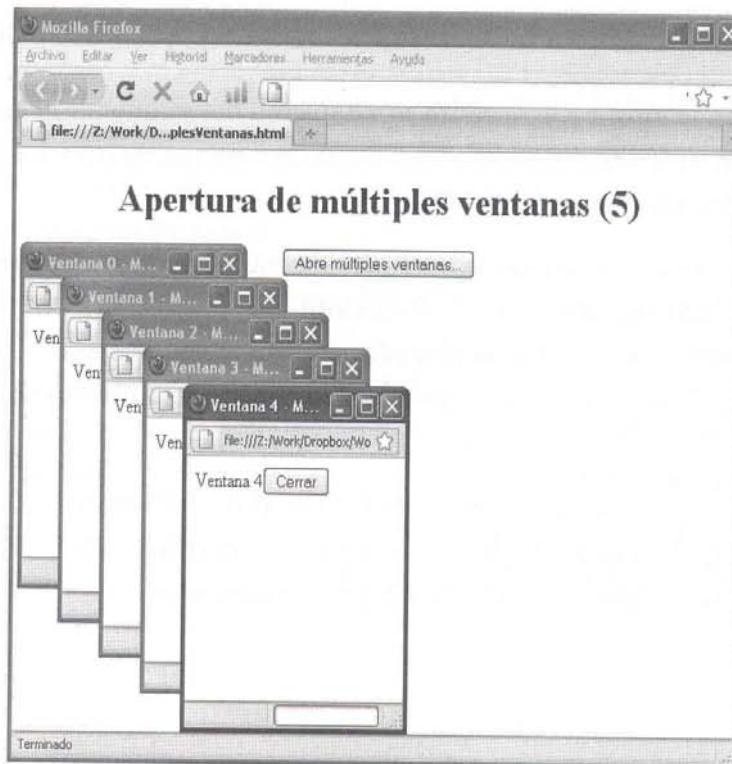


Figura 3.6. Apertura de múltiples ventanas

3.5.2 APARIENCIA DE LAS VENTANAS

Las ventanas cuentan con propiedades que podemos modificar con JavaScript. Estas propiedades permiten decidir el tamaño, la ubicación o los elementos que tendrá la ventana. La Tabla 3.20 contiene las principales propiedades que definen la apariencia de la ventana.

Tabla 3.20 Propiedades relacionadas con la apariencia de la ventana

Propiedad	Descripción
directories	Corresponde a los botones del directorio estándar del navegador.
height	Corresponde a la altura de la ventana.
menubar	Corresponde a la barra del menú.
resizable	Corresponde a la opción de cambiar o no el tamaño de la ventana.
scrollbars	Corresponde a las barras de desplazamiento.
status	Corresponde a la barra de estado.
toolbar	Corresponde a la barra de herramientas.
width	Corresponde a la anchura de la ventana.

Las propiedades que definen la altura y la anchura se establecen determinando el tamaño en píxeles. Las demás propiedades sirven para decidir si mostrar o no un elemento de la ventana, así que los posibles valores que pueden tomar estas últimas propiedades son uno (1) o cero (0).

La ubicación de una nueva ventana la determina automáticamente el navegador. Sin embargo, podemos establecer la ubicación exacta de las ventanas que abramos con JavaScript. Las propiedades `left` y `top` permiten definir esta ubicación y corresponden respectivamente a las coordenadas *x* e *y* en píxeles respecto a la esquina superior izquierda de la ventana. Los valores que definamos en estas propiedades debemos especificarlos en el tercer parámetro del método `open()`.

El siguiente ejemplo muestra una página web con un botón que permite la creación de una nueva ventada ubicada en la esquina superior izquierda de la pantalla. Esto ocurre ya que las propiedades `top` y `left` las ponemos iguales a cero. Además, establecemos una altura y anchura de 400 píxeles cada una.

```

<html>
  <head></head>
  <body>
    <center><h1> Apariencia de las ventanas</h1>
    <br>
    <input type="Button" value="Abre una ventana" onclick="
      myWindow1=window.open('', 'Nueva Ventana', 'top=0,
      left=0, width=400, height=400')
      myWindow1.document.write('<html>')
      myWindow1.document.write('<head>')
      myWindow1.document.write('<title>Ubicar una ventana
      </title>')
      myWindow1.document.write('</head>')
      myWindow1.document.write('<body>')
    " >
  </body>
</html>

```

```

myWindow1.document.write('top=0 <br> left=0 <br>
width=400 <br> height=400')
myWindow1.document.write('</body>')
myWindow1.document.write('</html>')
"/>
</center>
</body>
</html>

```

Debido a las diferentes resoluciones de pantalla de los usuarios, no siempre se consigue el efecto deseado cuando definimos la ubicación de las nuevas ventanas. Para evitar este inconveniente, podemos especificar posiciones relativas utilizando el objeto Screen y averiguar previamente la resolución de la pantalla con las propiedades screen.height y screen.width. Una vez conozcamos los datos de la resolución, podemos utilizar porcentajes de estos mismos para definir las propiedades left y top.

3.5.3 COMUNICACIÓN ENTRE VENTANAS

En apartados anteriores del libro hemos visto cómo desde una ventana se pueden abrir o cerrar nuevas ventanas. La primera se denomina ventana principal, mientras que las segundas se denominan ventanas secundarias. La comunicación e interacción entre estas ventanas no es un proceso del todo bidireccional. Por ejemplo, una ventana principal puede abrir y cerrar la secundaria, pero por motivos de seguridad, no es posible realizar lo contrario.

Las nuevas ventanas se declaran como objetos y se les asigna un nombre. Gracias a este nombre, desde la ventana principal podemos acceder a ella y establecer una comunicación e interacción con sus elementos y propiedades, además de aplicar métodos de JavaScript, tal y como hemos visto anteriormente con el método close(). La ventana secundaria tiene el atributo window.opener, el cual hace referencia a la ventana principal. De este modo, desde la ventana secundaria podemos acceder a los métodos y propiedades de la ventana principal.

En el siguiente ejemplo mostramos cómo acceder a la propiedad location del objeto Window de una ventana secundaria. Esta propiedad contiene la URL del documento activo. La ventana principal cuenta con un formulario que presenta un campo de texto en el que podemos escribir una URL, además de un botón que al presionarlo cambia la propiedad location de la ventana secundaria. Esta última ventana mostrará la URL que haya escrito el usuario.

```

<html>
<head></head>
<body>
<script>
var ventanaSecundaria = window.open("", "ventanaSec",
"width=500, height=500");
</script>
<center><h1> Comunicaci&acute;n entre ventanas </h1>
<br>
<form name=formulario>
<input type=text name=url size=50
value="http://www.">

```

```
<input type=button value="Mostrar URL en ventana  
secundaria" onclick=" ventanaSecundaria.location =  
document.formulario.url.value;">  
</form>  
</center>  
</body>  
</html>
```

ACTIVIDADES 3.5

- Abra el fichero Actividad-3.5a-ComunicaciónVentanas.html y observe que desde la ventana secundaria se envía información a la principal. Modifique este fichero para que el envío sea bidireccional. Es decir, que la información que se escriba en la ventana principal, se envíe a la ventana secundaria.



RESUMEN DEL CAPÍTULO

En este capítulo hemos presentado una completa descripción de los principales objetos nativos del lenguaje JavaScript. Además de estos objetos, hemos descrito otros que están relacionados directamente con las características y funcionamiento del navegador. De cada uno de estos objetos hemos mostrado sus principales propiedades y métodos.

A continuación, hemos expuesto el modo de generar elementos HTML desde código JavaScript. Además, hemos introducido el uso de los marcos HTML y la posibilidad de crear una interacción entre ellos gracias a JavaScript. Estos dos últimos temas nos permiten comprender los primeros pasos para poder generar páginas web dinámicas.

Por último, hemos introducido la gestión de las ventanas de un navegador. JavaScript permite abrir y cerrar una o más ventanas, modificar su apariencia gráfica y establecer una comunicación entre ellas.



EJERCICIOS PROPUESTOS

- 1. Cree una página web con un *script* que calcule los milisegundos que han pasado desde las 00:00 del 1 de enero del 2000 hasta la fecha actual.
- 2. Cree una página web que solicite una cadena de texto al usuario y devuelva la longitud de dicha cadena.
- 3. Cree un *script* que recoja el valor de la anchura y la altura total de la pantalla del usuario y calcule su diagonal. Recuerde: $diagonal = \sqrt{anchura^2 + altura^2}$.
- 4. Cree una página web que establezca las siguientes propiedades del objeto Document: color del texto = blanco, color de fondo = gris y título del documento = "Modificaciones del objeto Document".



TEST DE CONOCIMIENTOS

1 ¿El método `window.open()` necesita argumentos para poder abrir una nueva ventana?

- a) Verdadero.
- b) Falso.

2 ¿Qué propiedades se deben establecer para controlar el tamaño de una ventana?

- a) `top` y `left`.
- b) `frame` y `frameborder`.
- c) `width` y `height`.
- d) `resizable` y `size`.

3 El número devuelto al utilizar el método `getTime()` del objeto Date corresponde a:

- a) Días.
- b) Segundos.
- c) Milisegundos.
- d) Horas.

4 Se puede obtener el valor de la constante Pi mediante:

- a) `Math.3,14`.
- b) `Math.pi()`.
- c) `Math.PI`.
- d) Ninguna de las anteriores.

5 Se puede crear una ventana que no contenga las barras de desplazamiento si se establece:

- a) `toolbar = '0'`.
- b) `scrollbar = '1'`.
- c) `scrollbar = '0'`.
- d) `toolbar = '1'`.

6 ¿Qué método del objeto Window sirve para solicitar un valor al usuario?

- a) `alert()`.
- b) `prompt()`.
- c) `open()`.
- d) `print()`.

7

¿Cómo se puede ocultar el borde de un marco?

- a) frameborder = 'hide'.
- b) <frame> = '0'.
- c) frame = '1'.
- d) frameborder = '0'.

8

¿Cómo se define el número de marcos presente en un conjunto de marcos?

- a) Definiendo el valor frame.
- b) Definiendo el valor frameset.
- c) Definiendo el valor row.
- d) Definiendo el valor cols.

9

¿Cómo se cierra una ventana primaria desde una venta secundaria?

- a) Con el método window.close().
- b) No se puede cerrar una ventana primaria desde una secundaria.
- c) Con opener.close().
- d) Con onclick = "close()".

10

¿La comunicación entre ventanas es un proceso completamente bidireccional?

- a) Verdadero.
- b) Falso.