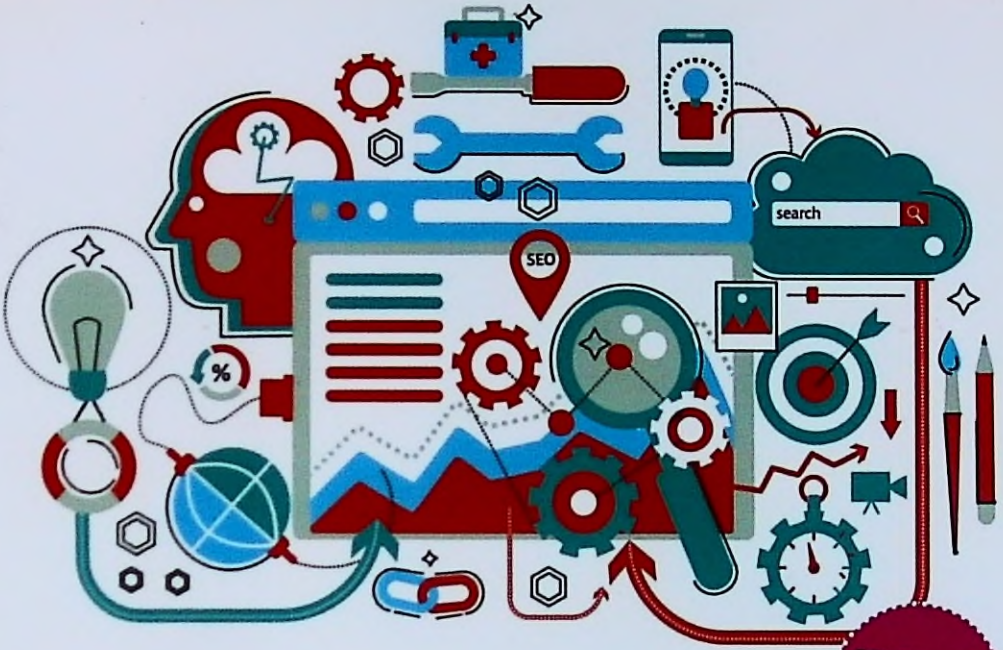

CONTENIDOS WEB

El gran libro de HTML5, CSS3 y JavaScript

J.D Gauchat

3ª Edición

26539 lelos de Caja Tradicional y Flexible
ño Web Adaptable
o y Audio

Marcombo

El gran libro de HTML5, CSS3 y JavaScript

El gran libro de HTML5, CSS3 y JavaScript

3a edición

J.D Gauchat

Marcombo

Edición original publicada en inglés por Mink Books con el título: *HTML5 for Masterminds*, © J.D Gauchat 2017.

Título de la edición en español:

El gran libro de HTML5, CSS3 y JavaScript

Tercera edición en español, año 2017

Gran Vía de les Corts Catalanes, 594
08007 Barcelona
www.marcombo.com

ISBN: 978-84-267-2463-2

D.L.: B-12571-2017

Printed in Spain

Tabla de contenidos

Introducción

Internet se ha convertido en una parte esencial de nuestras vidas y la Web es la pieza central que conecta todas las tecnologías involucradas. Desde noticias y entretenimientos hasta aplicaciones móviles y videojuegos, todo gira en torno a la Web. Debemos acceder a un sitio web para abrir una cuenta por cada servicio que usamos, para conectar nuestras aplicaciones y dispositivos móviles entre sí, o para compartir la puntuación alcanzada en nuestro juego preferido. La Web es el centro de operaciones de nuestra actividad diaria, y HTML5 es lo que lo ha hecho posible.

Todo comenzó tiempo atrás con una versión simplificada de un lenguaje de programación llamado *HTML*. El lenguaje, junto con identificadores y protocolos de comunicación, se concibió con el propósito de ofrecer la base necesaria para la creación de la Web. El propósito inicial del HTML era estructurar texto para poder compartir documentos entre ordenadores remotos. Con el transcurso del tiempo, la introducción de mejores sistemas y pantallas de color obligaron al lenguaje a evolucionar y poder así trabajar con otros medios además de texto, como imágenes y tipos de letras personalizados. Esta expansión complicó el trabajo de los desarrolladores, a quienes les resultaba cada vez más difícil crear y mantener sitios web extensos usando solo HTML. El problema se resolvió con la incorporación de un nuevo lenguaje llamado CSS, el cual permite a los desarrolladores preparar el documento que se va a presentar en pantalla.

La asociación entre HTML y CSS simplificó el trabajo de los desarrolladores, pero la capacidad de estos lenguajes para responder al usuario o realizar tareas como la reproducción de vídeo o audio era aún muy limitada. Al principio, algunas compañías independientes ofrecieron sus propias alternativas. Los lenguajes de programación como Java y Flash se volvieron muy populares, pero resultaron incapaces de ofrecer una solución definitiva. Las herramientas producidas con estas tecnologías aún operaban desconectadas del contenido y solo compartían con el documento un espacio en la pantalla. Esta débil asociación allanó el camino para la evolución de un lenguaje que ya se encontraba incluido en los navegadores y que, por lo tanto, estaba fuertemente integrado en HTML. Este lenguaje, llamado *JavaScript*, permitía a los desarrolladores acceder al contenido del documento y modificarlo de forma dinámica, solicitar datos adicionales desde el servidor, procesar información y mostrar los resultados en la pantalla, convirtiendo los sitios web en pequeñas aplicaciones. Originalmente, el rendimiento de los navegadores no era lo suficientemente bueno como para realizar algunas de estas tareas, pero con la incorporación de mejores intérpretes, los desarrolladores encontraron formas de aprovechar las capacidades de este lenguaje y comenzaron a crear aplicaciones útiles, confirmando a JavaScript como la mejor opción para complementar HTML y CSS.

Con la combinación de HTML, CSS y JavaScript, las tecnologías requeridas para construir la Web de las que disfrutamos hoy en día estaban listas, pero todavía existía un problema que resolver. Estos lenguajes habían sido desarrollados de forma independiente y, por lo tanto, seguían sus propios caminos, ajenos a los cambios presentados por los demás. La solución surgió con la definición de una nueva especificación llamada *HTML5*. HTML5 unifica todas las tecnologías involucradas en el desarrollo web. A partir de ahora, HTML se encarga de definir la estructura del documento, CSS prepara esa estructura y su contenido para ser mostrado en pantalla, y JavaScript introduce la capacidad de procesamiento necesaria para construir aplicaciones web completamente funcionales.

La integración entre HTML, CSS y JavaScript bajo el amparo de HTML5 cambió la Web para siempre. De la noche a la mañana se crearon nuevas compañías basadas en aplicaciones web y mercados completos, lo que originó una era de oro para el desarrollo web.

Implementando estas tecnologías, las oportunidades son infinitas. La Web está aquí para quedarse y tú puedes ser parte de ella.

IMPORTANTE: en el momento de escribir este libro, la mayoría de los navegadores admite HTML5, pero algunos aún presentan limitaciones. Por este motivo, le recomendamos ejecutar los ejemplos del libro en las últimas versiones de Google Chrome y Mozilla Firefox (**www.google.com/chrome** y **www.mozilla.com**). Si lo necesita, puede consultar el estado de la implementación de estas tecnologías en **www.caniuse.com**. En la parte inferior de la primera página del libro encontrará el código de acceso que le permitirá acceder de forma gratuita a los contenidos adicionales del libro en **www.marcombo.info**.

Capítulo 1
Desarrollo web

1.1 Sitios web

Los sitios web son archivos que los usuarios descargan con sus navegadores desde ordenadores remotos. Cuando un usuario decide acceder a un sitio web, le comunica al navegador la dirección del sitio y el programa descarga los archivos, procesa su contenido y lo muestra en pantalla.

Debido a que los sitios webs son de acceso público e Internet es una red global, estos archivos deben estar siempre disponibles. Por este motivo, los sitios web no se almacenan en ordenadores personales, sino en ordenadores especializados diseñados para despachar estos archivos a los usuarios que los solicitan. El ordenador que almacena los archivos y datos de un sitio web se llama *servidor* y el ordenador que accede a esta información se llama *cliente*, tal como lo ilustra la Figura 1-1.

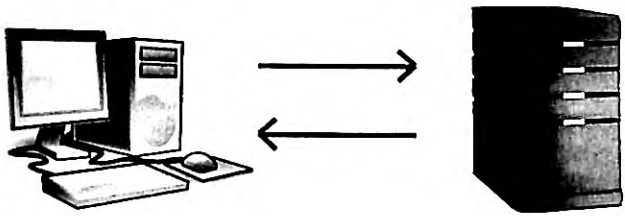

Figura 1-1: Clientes y servidores

Los servidores son muy similares a los ordenadores personales, con la diferencia de que están continuamente conectados a la red y ejecutando programas que les permiten responder a las solicitudes de los usuarios, sin importar cuándo se reciben o de donde proceden. Los programas más populares para servidores son Apache, para sistemas Linux, e IIS (Internet Information Server), creado por Microsoft para sistemas Windows. Entre otras funciones, estos programas son responsables de establecer la conexión entre el cliente y el servidor, controlar el acceso de los usuarios, administrar los archivos, y despachar los documentos y recursos requeridos por los clientes.

Archivos

Los sitios web están compuestos de múltiples documentos que el navegador descarga cuando el usuario los solicita. Los documentos que conforman un sitio web se llaman *páginas* y el proceso de abrir nuevas páginas *navegar* (el usuario navega a través de las páginas del sitio). Para desarrollar un sitio web, tenemos que crear un archivo por cada página que queremos incluir. Junto con estos archivos, también debemos incluir los archivos con las imágenes y cualquier otro recurso que queremos mostrar dentro de estas páginas (las imágenes y otros

medios gráficos se almacenan en archivos aparte). En la Figura 1-2 se representa cómo se muestran los directorios y archivos de un sitio web una vez que se suben al servidor.

***Figura 1-2:** Archivos de un sitio web*

En el ejemplo de la Figura 1-2 se incluyen dos directorios llamados *imagenes* y *recursos*, y tres archivos llamados *contacto.html*, *index.html* y *news.html*. Los directorios se crearon para almacenar las imágenes que queremos mostrar dentro de las páginas web y otros recursos, como los archivos que contienen los códigos en CSS y JavaScript. Por otro lado, los archivos de este ejemplo representan las tres páginas web que queremos incluir en este sitio. El archivo index.html contiene el código y la información correspondiente a la página principal (la página que el usuario ve cuando entra a nuestro sitio web por primera vez), el archivo contacto.html contiene el código necesario para presentar un formulario que el usuario puede rellenar para enviarnos un mensaje y el archivo noticias.html contiene el código necesario para mostrar las noticias que queremos compartir con nuestros usuarios. Cuando un usuario accede a nuestro sitio web por primera vez, el navegador descarga el archivo index.html y muestra su contenido en la ventana. Si el usuario realiza una acción para ver las noticias ofrecidas por nuestro sitio web, el navegador descarga el archivo noticias.html desde el servidor y reemplaza el contenido del archivo index.html por el contenido de este nuevo archivo. Cada vez que el usuario quiere acceder a una nueva página web, el navegador tiene que descargar el correspondiente archivo desde el servidor, procesarlo y mostrar su contenido en la pantalla.

Los archivos de un sitio web son iguales que los archivos que podemos encontrar en un ordenador personal. Todos tiene un nombre seleccionado por el desarrollador y una extensión que refleja el lenguaje usado para programar su contenido (en nuestro ejemplo, los archivos tienen la extensión .html porque fueron programados en HTML). Aunque podemos asignar cualquier nombre que queramos a estos archivos, el archivo que genera la página inicial presenta algunos requisitos. Algunos servidores como Apache designan archivos por defecto en caso de que el usuario no especifique ninguno. El nombre utilizado con más frecuencia es *index*. Si un usuario accede al servidor sin especificar el nombre del archivo que intenta abrir, el servidor busca un archivo con el nombre *index* y lo envía de vuelta al cliente. Por esta razón, el archivo index es el punto de entrada de nuestro sitio web y siempre debemos incluirlo.

IMPORTANTE: los servidores son flexibles en cuanto a los nombres que podemos asignar a nuestros archivos, pero existen algunas reglas que debería seguir para asegurarse de que sus archivos son accesibles. Evite usar espacios. Si necesita separar palabras use el guion bajo en su lugar (_). Además, debe considerar que algunos caracteres realizan funciones específicas en la Web, por lo que es mejor evitar caracteres especiales como ?, %, #, /, y usar solo letras minúsculas sin acentos y números.

Lo básico: aunque *index* es el nombre más común, no es el único que podemos asignar al archivo por defecto. Algunos servidores designan otros nombres como *home* o *default*, e incluyen diferentes extensiones. Por ejemplo, si en lugar de programar nuestros documentos en HTML lo hacemos en un lenguaje de servidor como PHP, debemos asignar a nuestro archivo index el nombre *index.php*. El servidor contiene una lista de archivos y continúa buscando hasta que encuentra uno que coincida con esa lista. Por ejemplo, Apache primero busca por un archivo con el nombre index y la extensión .html, pero si no lo encuentra, busca por un archivo con el nombre index y la extensión .php. Estudiaremos HTML y PHP más adelante en este y otros capítulos.

Dominios y URL

Los servidores se identifican con un valor llamado *IP* (Internet Protocol). Esta IP es única para cada ordenador y, por lo tanto, trabaja como una dirección que permite ubicar a un ordenador dentro de una red. Cuando el navegador tiene que acceder a un servidor para descargar el documento solicitado por el usuario, primero busca el servidor a través de esta dirección IP y luego le pide que le envíe el documento.

Las direcciones IP están compuestas por números enteros entre 0 y 255 separados por un punto, o números y letras separadas por dos puntos, dependiendo de la versión (IPv4 o IPv6). Por ejemplo, la dirección 216.58.198.100 corresponde al servidor donde se encuentra alojado el sitio web de Google. Si escribimos esta dirección IP en la barra de navegación de nuestro navegador, la página inicial de Google se descarga y muestra en pantalla.

En teoría, podríamos acceder a cualquier servidor utilizando su dirección IP, pero estos valores son crípticos y difíciles de recordar. Por esta razón, Internet utiliza un sistema que identifica a cada servidor con un nombre específico. Estos nombres personalizados, llamados *dominios*, son identificadores sencillos que cualquier persona puede recordar, como *google* o *yahoo*, con una extensión que determina el propósito del sitio web al que hacen referencia, como *.com* (comercial) o *.org* (organización). Cuando el usuario le pide al navegador que acceda al sitio web con el dominio www.google.com, el navegador accede primero a un servidor llamado *DNS* que contiene una lista de dominios con sus respectivas direcciones IP. Este servidor encuentra la IP 216.58.198.100 asociada al dominio www.google.com, la devuelve al navegador, y entonces el navegador accede al sitio web de Google por medio de esta IP. Debido a que las direcciones IP de los sitios web siempre se encuentran asociadas a sus dominios, no necesitamos recordar la dirección de un servidor para acceder a él, solo tenemos que recordar el domino y el navegador se encarga de encontrar el servidor y descargar los archivos por nosotros.

Los sitios web están compuestos por múltiples archivos, por lo que debemos agregar el nombre del archivo al dominio para indicar cuál queremos descargar. Esta construcción se llama *URL* e incluye tres partes, tal como se describe en la Figura 1-3.

Figura 1-3: URL

La primera parte de la URL es una cadena de caracteres que representa el protocolo de comunicación que se utilizará para acceder al recurso (el protocolo creado para la Web se llama *HTTP*), el siguiente componente es el dominio del sitio web y el último componente es el nombre del recurso que queremos descargar (puede ser un archivo, como en nuestro ejemplo, o una ruta a seguir que incluye el directorio donde el archivo se encuentra almacenado (por ejemplo, http://www.ejemplo.com/imagenes/milogo.jpg). La URL en nuestro ejemplo le pide al navegador que utilice el protocolo HTTP para acceder al archivo contacto.html, ubicado en el servidor identificado con el domino www.ejemplo.com.

Las URL se utilizan para ubicar cada uno de los documentos en el sitio web y son, por lo tanto, necesarias para navegar por el sitio. Si el usuario no especifica ningún archivo, el servidor devuelve el archivo por defecto, pero de ahí en adelante, cada vez que el usuario realiza una acción para abrir una página diferente, el navegador debe incluir en la URL el nombre del archivo que corresponde a la página solicitada.

IMPORTANTE: una vez que ha conseguido el dominio para su sitio web, puede crear subdominios. Los subdominios son enlaces directos a directorios y nos permiten crear múltiples sitios web en una misma cuenta. Un subdominio se crea con el nombre del directorio y el dominio conectados por un punto. Por ejemplo, si su dominio es www.ejemplo.com y luego crea un subdominio para un directorio llamado *recursos*, podrá acceder directamente al directorio escribiendo en el navegador la URL http://recursos.ejemplo.com.

Lo básico: existen diferentes protocolos que los ordenadores utilizan para comunicarse entre ellos, y transferir recursos y datos. HTTP (HyperText Transfer Protocol) es el protocolo de comunicación que se utiliza para acceder a documentos web. Siempre tenemos que incluir el prefijo HTTP en la URL cuando el recurso al que estamos tratando de acceder pertenece a un sitio web, pero en la práctica esto no es necesario porque los navegadores lo hacen de forma automática. Existe otra versión disponible de este protocolo llamado *HTTPS*. La S indica que la conexión es encriptada por protocolos de encriptación como TLS o SSL. Los sitios web pequeños no necesitan encriptación, pero se recomienda utilizarla en sitios web que manejan información sensible.

Hipervínculos

En teoría, podemos acceder a todos los documentos de un sitio web escribiendo la URL en la barra de navegación del navegador. Por ejemplo, si queremos acceder a la página inicial en español del sitio web *For Masterminds*, podemos insertar la URL http://www.formasterminds.com/esindex.php, o bien insertar la URL http://www.formasterminds.com/escontact.php para abrir la página que nos permite enviar un mensaje a su desarrollador. Aunque es posible acceder a todos los archivos del sitio web usando este método, no es práctico. En primer lugar, los usuarios no conocen los nombres que el desarrollador eligió para cada archivo y, por lo tanto, estarán limitados a aquellos nombres que pueden adivinar o solo a la página principal que devuelve por defecto. En segundo lugar, los sitios web pueden estar compuestos por docenas o incluso miles de páginas web (algunos

sitios contienen millones) y la mayoría de los documentos serían imposibles de encontrar. La solución se encontró con la definición de hipervínculos. Los hipervínculos, también llamados *enlaces*, son referencias a documentos dentro de las páginas de un sitio web. Incorporando estos enlaces, una página puede contener referencias a otras páginas. Si el usuario hace clic con el ratón en un enlace, el navegador sigue esa referencia y el documento indicado por la URL de la referencia se descarga y muestra en pantalla. Debido a estas conexiones entre páginas, los usuarios pueden navegar en el sitio web y acceder a todos sus documentos simplemente haciendo clic en sus enlaces.

Lo básico: los enlaces son lo que transforma a un grupo de archivos en un sitio web. Para crear un sitio web, debe programar los documentos correspondientes a cada página e incluir dentro de las mismas los enlaces que establecen una ruta que el usuario puede seguir para acceder a cada una de ellas. Estudiaremos cómo incorporar enlaces en nuestros documentos en el Capítulo 2.

URL absolutas y relativas

Los hipervínculos son procesados por el navegador antes de ser usados para acceder a los documentos. Por esta razón, se pueden definir con URL absolutas o relativas. Las URL absolutas son aquellas que incluyen toda la información necesaria para acceder al recurso (ver Figura 1-3), mientras que las relativas son aquellas que solo declaran la parte de la ruta que el navegador tiene que agregar a la URL actual para acceder al recurso. Por ejemplo, si tenemos un hipervínculo dentro de un documento que referencia una imagen dentro del directorio imagenes, podemos crear el enlace con la URL http://www.ejemplo.com/imagenes/miimagen.png, pero también tenemos la opción de declararla como "imagenes/miimagen.png" y el navegador se encargará de agregar a esta ruta la URL actual y descargar la imagen.

Las URL relativas no solo pueden determinar una ruta hacia abajo, sino también hacia arriba de la jerarquía. Por ejemplo, si tenemos un documento dentro del directorio recursos en el ejemplo de la Figura 1-2 y queremos acceder a un documento en el directorio raíz, podemos crear una URL relativa usando los caracteres ../ al comienzo de la ruta. Si el documento que queremos acceder es noticias.html, la URL relativa sería ../noticias.html. Los dos puntos .. le indican al navegador que el documento al que queremos acceder se encuentra dentro del directorio padre del actual directorio (recursos, en nuestro ejemplo).

1.2 Lenguajes

Como mencionamos en la introducción, HTML5 incorpora tres características (estructura, estilo, y funcionalidad), con lo que integra tres lenguajes de programación independientes: HTML, CSS, y JavaScript. Estos lenguajes están compuestos por grupos de instrucciones que los navegadores pueden interpretar para procesar y mostrar los documentos al usuario. Para crear nuestros documentos, tenemos que aprender todas las instrucciones incluidas en estos lenguajes y saber cómo organizarlas.

HTML

HTML (*HyperText Markup Language*) es un lenguaje compuesto por un grupo de etiquetas definidas con un nombre rodeado de paréntesis angulares. Los paréntesis angulares delimitan la etiqueta y el nombre define el tipo de contenido que representa. Por ejemplo, la etiqueta `<html>` indica que el contenido es código HTML. Algunas de estas etiquetas son declaradas individualmente (por ejemplo, `<br>`) y otras son declaradas en pares, que incluyen una de apertura y otra de cierre, como `<html></html>` (en la etiqueta de cierre el nombre va precedido por una barra invertida). Las etiquetas individuales y las de apertura pueden incluir atributos para ofrecer información adicional acerca de sus contenidos (por ejemplo, `<html lang="es">`). Las etiquetas individuales y la combinación de etiquetas de apertura y cierre se llaman *elementos*. Los elementos compuestos por una sola etiqueta se usan para modificar el contenido que los rodea o incluir recursos externos, mientras que los elementos que incluyen etiquetas de apertura y cierre se utilizan para delimitar el contenido del documento, tal como ilustra la Figura 1-4.

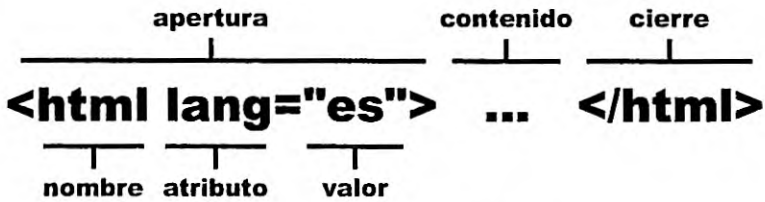

Figura 1-4: Elemento HTML

Se deben combinar múltiples elementos para definir un documento. Los elementos son listados en secuencia de arriba abajo y pueden contener otros elementos en su interior. Por ejemplo, el elemento `<html>` que se muestra en la Figura 1-4 declara que su contenido debe ser interpretado como código HTML. Por lo tanto, el resto de los elementos que describen el contenido de ese documento se deben declarar entre las etiquetas `<html>` y `</html>`. A su vez, los elementos dentro del elemento `<html>` pueden incluir otros elementos. El siguiente ejemplo muestra un documento HTML sencillo que incluye todos los elementos necesarios para definir una estructura básica y mostrar el mensaje HOLA MUNDO! en la pantalla.

```
<!DOCTYPE html>
<html lang="es">
  <head>
    <title>Mi primer documento HTML</title>
  </head>
  <body>
    <p>HOLA MUNDO!</p>
  </body>
</html>
```

Listado 1-1: Creando un documento HTML

En el ejemplo del Listado 1-1 presentamos un código sencillo, pero con una estructura compleja. En la primera línea, se encuentra una etiqueta individual que declara el tipo de documento (`<!DOCTYPE html>`) seguida por una etiqueta de apertura `<html lang="es">`. Entre las etiquetas `<html>` y `</html>` se incluyen otros elementos que representan la cabecera y el cuerpo del documento (`<head>` y `<body>`), los cuales a su vez encierran más elementos con sus respectivos contenidos (`<title>` y `<p>`), demostrando cómo se compone un documento HTML. Los elementos se listan uno a continuación de otro y también dentro de otros elementos, de modo que se construye una estructura de tipo árbol con el elemento `<html>` como raíz.

Lo básico: en general, todo elemento puede ser anidado, convertirse en un contenedor o ser contenido por otros elementos. Los elementos exclusivamente estructurales como `<html>`, `<head>` y `<body>` tienen un lugar específico en un documento HTML, pero el resto son flexibles, tal como veremos en el Capítulo 2.

Como ya mencionamos, las etiquetas individuales y de apertura pueden incluir atributos. Por ejemplo, la etiqueta de apertura `<html>` declarada en el Listado 1-1 no está solo compuesta por el nombre `html` y los paréntesis angulares, sino también por el texto `lang="es"`. Este es un atributo con un valor. El nombre del atributo es `lang` y el valor `es` se asigna al atributo usando el carácter `=`. Los atributos ofrecen información adicional acerca del elemento y su contenido. En este caso, el atributo `lang` declara el idioma del contenido del documento (`es` por Español).

Lo básico: los atributos se declaran siempre dentro de la etiqueta de apertura (o etiquetas individuales) y pueden tener una estructura que incluye un nombre y un valor, como el atributo `lang` de la etiqueta `<html>`, o representar un valor por sí mismos, como el atributo `html` de la etiqueta `<!DOCTYPE>`. Estudiaremos los elementos HTML y sus atributos en el Capítulo 2.

CSS

CSS (*Cascading Style Sheets*) es el lenguaje que se utiliza para definir los estilos de los elementos HTML, como el tamaño, el color, el fondo, el borde, etc. Aunque todos los navegadores asignan estilos por defecto a la mayoría de los elementos, estos estilos generalmente están lejos de lo que queremos para nuestros sitios web. Para declarar estilos personalizados, CSS utiliza propiedades y valores. Esta construcción se llama *declaración* y su sintaxis incluye dos puntos después del nombre de la propiedad, y un punto y coma al final para cerrar la línea.

Figura 1-5: Propiedad CSS

En el ejemplo de la Figura 1-5, el valor `#FF0000` se asigna a la propiedad `color`. Si esta propiedad se aplica luego a un elemento HTML, el contenido de ese elemento se mostrará en color rojo (el valor `#FF0000` representa el color rojo).

Las propiedades CSS se pueden agrupar usando llaves. Un grupo de una o más propiedades se llama *regla* y se identifica por un nombre llamado *selector*.

```
body {
  width: 100%;
  margin: 0px;
  background-color: #FF0000;
}
```

***Listado 1-2:** Declarando reglas CSS*

El Listado 1-2 declara una regla con tres propiedades: **width**, **margin** y **background-color**. Esta regla se identifica con el nombre **body**, lo que significa que las propiedades serán aplicadas al elemento **<body>**. Si incluimos esta regla en un documento, el contenido del documento se extenderán hacia los límites de la ventana del navegador y tendrán un fondo rojo.

Lo básico: existen diferentes técnicas para aplicar estilos CSS a elementos HTML. Estudiaremos las propiedades CSS y cómo incluirlas en un documento HTML en los Capítulos 3 y 4.

JavaScript

A diferencia de HTML y CSS, JavaScript es un lenguaje de programación. Para ser justos, todos estos lenguajes pueden ser considerados lenguajes de programación, pero en la práctica existen algunas diferencias en la forma en la que suministran las instrucciones al navegador. HTML es como un grupo de indicadores que el navegador interpreta para organizar la información, CSS puede ser considerado como una lista de estilos que ayudan al navegador a preparar el documento para ser presentado en pantalla (aunque la última especificación lo convirtió en un lenguaje más dinámico), pero JavaScript es un lenguaje de programación, comparable con cualquier otro lenguaje de programación profesional como C++ o Java. JavaScript difiere de los demás lenguajes en que puede realizar tareas personalizadas, desde almacenar valores hasta calcular algoritmos complejos, incluida la capacidad de interactuar con los elementos del documento y procesar su contenido dinámicamente.

Al igual que HTML y CSS, JavaScript se incluye en los navegadores y, por lo tanto, se encuentra disponible en todos nuestros documentos. Para declarar código JavaScript dentro de un documento, HTML ofrece el elemento **<script>**. El siguiente ejemplo es una muestra de un código escrito en JavaScript.

```
<script>
  function cambiarColor() {
    document.body.style.backgroundColor = "#0000FF";
  }
  document.addEventListener("click", cambiarColor);
</script>
```

***Listado 1-3:** Declarando código JavaScript*

El código en el Listado 1-3 cambia el color de fondo del elemento **<body>** a azul cuando el usuario hace clic en el documento.

Lo básico: con el elemento **<script>** también podemos cargar código JavaScript desde archivos externos. Estudiaremos el elemento **<script>** en el Capítulo 2 y el lenguaje JavaScript en el Capítulo 6.

Lenguajes de servidor

Los códigos programados en HTML, CSS, y JavaScript son ejecutados por el navegador en el ordenador del usuario (el cliente). Esto significa que, después de que los archivos del sitio web se suben al servidor, permanecen inalterables hasta que se descargan en un ordenador personal y sus códigos son ejecutados por el navegador. Aunque esto permite la creación de sitios web útiles e interactivos, hay momentos en los cuales necesitamos procesar la información en el servidor antes de enviarla al usuario. El contenido producido por esta información se denomina *contenido dinámico*, y es generado por códigos ejecutados en el servidor y programados en lenguajes que fueron especialmente diseñados con este propósito (lenguajes de servidor). Cuando el navegador solicita un archivo que contiene este tipo de código, el servidor lo ejecuta y luego envía el resultado como respuesta al usuario. Estos códigos no solo se utilizan para generar contenido y documentos en tiempo real, sino también para procesar la información enviada por el navegador, almacenar datos del usuario en el servidor, controlar cuentas, etc.

Existen varios lenguajes disponibles para crear código ejecutable en los servidores. Los más populares son PHP, Ruby, y Python. El siguiente ejemplo es una muestra de un código escrito en PHP.

```
<?php
  $nombre = $_GET['minombre'];
  print('Su nombre es: '.$nombre);
?>
```

***Listado 1-4:** Declarando código ejecutable en el servidor*

El código del Listado 1-4 recibe un valor enviado por el navegador, lo almacena en la memoria y crea un mensaje con el mismo. Cuando se ejecuta este código, se crea un nuevo documento que contiene el mensaje final, el archivo se envía de vuelta al cliente y finalmente el navegador muestra su contenido en pantalla.

IMPORTANTE: los lenguajes de servidor utilizan su propia tecnología, pero trabajan junto con HTML5 para llevar un registro de las cuentas de usuarios, almacenar información en el servidor, manejar bases de datos, etc. El tema va más allá del propósito de este libro. Para obtener más información sobre cómo programar en PHP, Ruby, o Python, descargue los contenidos adiconales en **www.marcombo.info**.

1.3 Herramientas

Crear un sitio web involucra múltiples pasos. Tenemos que programar los documentos en HTML, crear los archivos con los estilos CSS y los códigos en JavaScript, configurar el servidor

que hará que el sitio sea visible a los usuarios y transferir todos los archivos desde nuestro ordenador al servidor. Por fortuna existen muchas herramientas disponibles que nos pueden ayudar con estas tareas. Estas herramientas son muy fáciles de usar y la mayoría se ofrecen de forma gratuita.

IMPORTANTE: en esta sección del capítulo introducimos todas las herramientas que necesitará para crear sus sitios web y ofrecerlos a sus usuarios. Esto incluye las herramientas requeridas para programar y diseñar un sitio web, pero también otras que necesitará para configurarlo y probarlo antes de hacerlo público. La mayoría de los ejemplos de este libro no tienen que subirse a un servidor para poder trabajar adecuadamente y, por lo tanto, puede ignorar parte de esta información hasta que sea requerida por sus proyectos.

Editores

Los documentos HTML, así como los archivos CSS y JavaScript, son archivos de texto, por lo que podemos usar cualquier editor incluido en nuestro ordenador para crearlos, como el bloc de notas de Windows o la aplicación editor de texto de los ordenadores de Apple, pero también existen editores de texto especialmente diseñados para programadores y desarrolladores web que pueden simplificar nuestro trabajo. Estos editores resaltan texto con diferentes colores para ayudarnos a identificar cada parte del código, o listan los archivos de un proyecto en un panel lateral para ayudarnos a trabajar con múltiples archivos al mismo tiempo. La siguiente es una lista de los editores y de IDE (Integrated Development Environments) más populares disponibles para ordenadores personales y ordenadores de Apple.

- **Atom** (www.atom.io) es un editor gratuito, simple de usar y altamente personalizable (recomendado).
- **Brackets** (www.brackets.io) es un editor gratuito creado por Adobe.
- **KompoZer** (www.kompozer.net) es un editor gratuito con un panel de vista previa que facilita la búsqueda de partes específicas del documento a modificar.
- **Aptana** (www.aptana.com) es una IDE gratuita con herramientas que simplifican la administración de archivos y proyectos.
- **NetBeans** (www.netbeans.org) es una IDE gratuita con herramientas para administrar archivos y proyectos.
- **Sublime** (www.sublimetext.com) es un editor de pago con una versión gratuita de evaluación.
- **Komodo** (www.komodoide.com) es una IDE de pago que puede trabajar con una cantidad extensa de lenguajes de programación.
- **Dreamweaver** (www.adobe.com/products/dreamweaver.html) es una IDE de pago con tecnología WYSIWYG incorporada (Lo Que Ves Es Lo Que Obtienes) que nos permite ver los resultados de la ejecución del código en tiempo real.

Trabajar con un editor es simple: tenemos que crear un directorio en nuestro disco duro donde vamos a almacenar los archivos del sitio web, abrir el editor, y crear dentro de este directorio todos los archivos y directorios adicionales que necesitamos para nuestro proyecto. La Figura 1-6 muestra cómo se ve el editor Atom cuando se abre por primera vez.

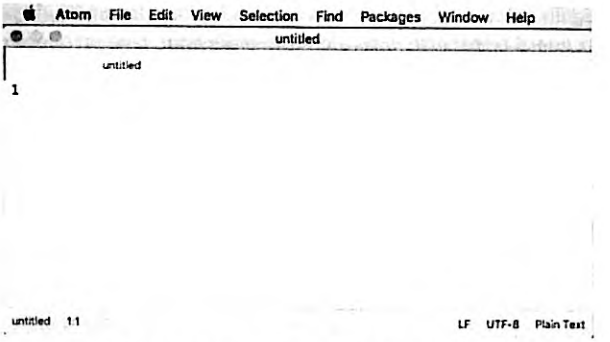

Figura 1-6: Editor Atom con un archivo vacío

Este editor tiene una opción en el menú **File** (Archivo) para abrir un proyecto (**Add Project Folder**). La opción se muestra en la Figura 1-7, número 1.

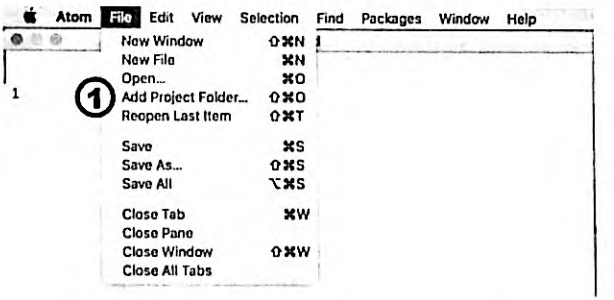

Figura 1-7: Opción para agregar un proyecto

Si hacemos clic en esta opción y luego seleccionamos el directorio creado para nuestro proyecto, el editor abre un nuevo panel a la izquierda con la lista de archivos dentro del directorio. La Figura 1-8, a continuación, muestra un directorio llamado **Test** creado para contener los archivos del ejemplo de la Figura 1-2.

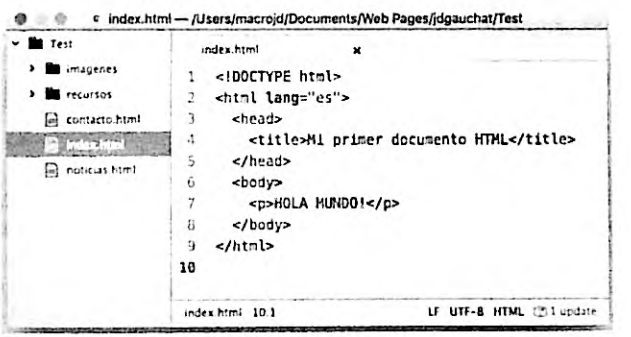

Figura 1-8: Archivos del proyecto

Hágalo usted mismo: visite **www.atom.io** para descargar el editor Atom. Una vez que el editor esté instalado en su ordenador, cree un nuevo directorio en su disco duro para almacenar los archivos de su sitio web. Abra Atom, vaya al menú **File** y seleccione la opción **Add Project Folder** (Figura 1-7, número 1). Seleccione el directorio que acaba de crear. Abra de nuevo el menú **File** y seleccione la opción **New File** (Nuevo Archivo). Copie el código del Listado 1-1 y grabe el archivo con el nombre index.html. Después de seleccionar el archivo en el panel de la izquierda, debería ver algo similar a lo que se muestra en la Figura 1-8.

IMPORTANTE: explicar cómo trabajar con Atom, u otro editor, va más allá del propósito de este libro, pero hemos incluido enlaces a cursos y vídeos en nuestro sitio web con más información. Para acceder a ellos, descargue los contenidos adicionales y haga clic en las opciones **Enlaces** y **Vídeos**.

Registro de dominios

Una vez que nuestro sitio web está listo para ser presentado en público, tenemos que registrar el dominio que los usuarios van a escribir en la barra de navegación para acceder a él. Como ya mencionamos, un domino es simplemente un nombre personalizado con una extensión que determina el propósito del sitio web. El nombre puede ser cualquiera que deseemos, y contamos con varias opciones para definir la extensión, desde extensiones con propósitos comerciales, como .com o .biz., a aquellas sin ánimo de lucro o personales, como .org, .net o .info, por no mencionar las extensiones regionales que incluyen un valor adicional para determinar la ubicación del sitio web, como .co.uk para sitios web en el Reino Unido o .eu para sitios web relacionados con la Union Europea.

Para obtener un domino para nuestro sitio web, tenemos que abrir una cuenta con un registrante y adquirirlo. La mayoría de los dominios requieren del pago de un arancel anual, pero el proceso es relativamente sencillo y hay muchas compañías disponibles que pueden hacerse cargo del trámite por nosotros. La más popular es GoDaddy (www.godaddy.com), pero la mayoría de las compañías que ofrecen servicios para desarrolladores también incluyen la posibilidad de registrar un dominio. Como dijimos, el proceso de registro es sencillo; tenemos que decidir el nombre y la extensión que vamos a asignar a nuestro dominio, realizar una búsqueda para asegurarnos de que el nombre que hemos elegido no está siendo utilizado y se encuentra disponible, y luego hacer el pedido (las compañías mencionadas con anterioridad facilitan todas las herramientas necesarias para este propósito).

Cuando el dominio está registrado, el sistema nos pide los nombres de servidores (nameservers) que queremos asociar al dominio. Estos nombres son cadenas de texto compuestas por un dominio y un prefijo, generalmente NS1 y NS2, que determinan la ubicación de nuestro sitio web (los nombres de servidor o nameservers los facilita el servidor en el que se almacena nuestro sitio web). Si aún no contamos con estos nombres, podemos usar los que ofrece la compañía y cambiarlos más adelante.

IMPORTANTE: la compañía que registra su dominio asigna nombres de servidor por defecto que ellos usan como destino provisional (también conocido como *aparcamiento* o *parking*). En principio puede asignar estos nombres y cambiarlos más adelante cuando su servidor esté listo. Algunas compañías ofrecen el registro de dominio junto con el alquiler de servidores y, por lo tanto, pueden encargarse de la configuración del dominio por nosotros si usamos sus servidores.

Alojamiento web

Configurar y mantener un servidor exige conocimientos que no todos los desarrolladores poseen. Por este motivo, existen compañías que ofrecen un servicio llamado alojamiento web (*web hosting*), que permite a cualquier individuo alquilar un servidor configurado y listo para almacenar, y operar uno o múltiples sitios web.

Existen diferentes tipos de alojamiento web disponible, desde aquellos que permiten que varios sitios web operen desde un mismo servidor (alojamiento compartido) hasta servicios más profesionales que reservan un servidor completo para un único sitio web (alojamiento dedicado), o distribuyen un sitio web extenso en muchos servidores (alojamiento en la nube), incluidas varias opciones intermedias.

La principal ventaja de tener una cuenta de alojamiento web es que todas ofrecen un panel de control con opciones para crear y configurar nuestro sitio web. Las siguientes son las opciones más comunes que nos encontraremos en la mayoría de estos servicios.

- **File Manager** es una herramienta web que nos permite administrar los archivos de nuestro sitio. Con esta herramienta podemos subir, bajar, editar o eliminar archivos en el servidor desde el navegador, sin tener que usar ninguna otra aplicación.
- **FTP Accounts** es un servicio que nos permite administrar las cuentas que usamos para conectarnos al servidor por medio de FTP. FTP (File Transfer Protocol) es un protocolo de comunicación diseñado para transferir archivos desde un ordenador a otro en la red.
- **MySQL Databases** es un servicio que nos permite crear bases de datos para nuestro sitio web.
- **phpMyAdmin** es una aplicación programada en PHP que podemos usar para administrar las bases de datos creadas para nuestro sitio web.
- **Email Accounts** es un servicio que nos permite crear cuentas de email con el dominio de nuestro sitio web (por ejemplo, info@midominio.com).

El panel de control más popular en el mercado es *cPanel*. La Figura 1-9 muestra el diseño y algunas de las opciones que ofrece.

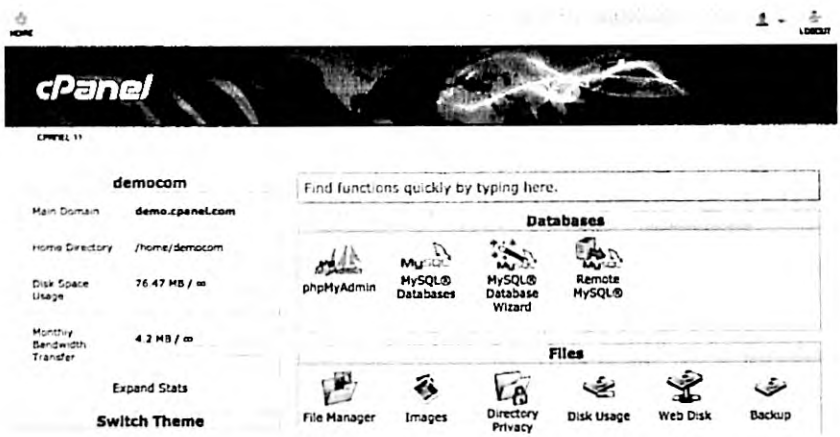

Figura 1-9: Opciones ofrecidas por cPanel

El coste de una cuenta de alojamiento puede variar entre algunos dólares por una cuenta compartida hasta cientos de dólares al mes por un servidor dedicado. Una vez que abrimos la cuenta, la compañía nos envía un email con la información que necesitamos para acceder al panel de control y configurar el servidor. El sistema de la compañía generalmente crea todas las cuentas básicas que necesitamos, incluida una cuenta FTP para subir los archivos, tal como veremos a continuación.

Lo básico: además de las cuentas de alojamiento de pago, existe el alojamiento gratuito que podemos usar para practicar, pero estos servicios incluyen propaganda o imponen restricciones que impiden el desarrollo de sitios web profesionales. Siempre se recomienda comenzar con una cuenta de alojamiento compartido que puede costar unos 5 dólares al mes para aprender cómo trabaja un servicio de alojamiento profesional y estudiar todas las opciones que ofrece. Varias compañías incluyen en sus servicios este tipo de alojamiento. Las más populares en este momento son www.godaddy.com y www.hostgator.com.

Programas FTP

Como acabamos de mencionar, las cuentas de alojamiento web ofrecen un servicio para administrar los archivos del sitio web desde el navegador. Esta es una página web a la que podemos acceder desde el panel de control para subir, bajar y editar los archivos en el servidor. Es una herramienta útil, pero solo práctica cuando necesitamos realizar pequeñas modificaciones o subir unos pocos archivos. La herramienta aprovecha un sistema que se encuentra integrado en los navegadores y que trabaja con un protocolo llamado *FTP* (File Transfer Protocol) usado para transferir archivos desde un ordenador a otro en una red. Los navegadores incluyen este sistema porque lo necesitan para permitir a los usuarios descargar archivos pero, debido a que su principal propósito es descargar y mostrar sitios web en la pantalla, ofrecen una mala experiencia a la hora de manipular estos archivos. Por esta razón, los desarrolladores profesionales no utilizan el navegador sino programas diseñados específicamente para transferir archivos entre un cliente y un servidor usando el protocolo FTP.

El mercado ofrece varios programas FTP, incluidas versiones de pago y gratuitas. El programa gratuito más popular se llama *Filezilla* y se encuentra disponible en **www.filezilla-project.org**. Este programa ofrece varios paneles con información acerca de la conexión y los ordenadores que participan, incluidos dos paneles lado a lado con la lista de los archivos locales y remotos que podemos transferir entre ordenadores con solo arrastrarlos de un panel a otro.

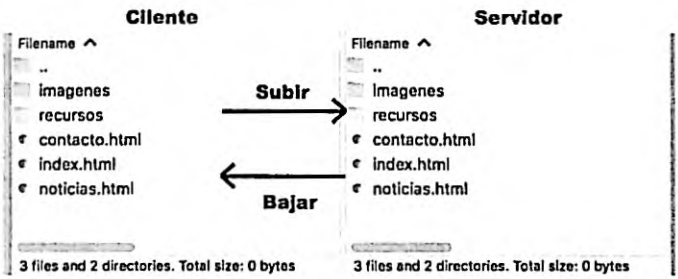

Figura 1-10: Interfaz de Filezilla

Cuando abrimos una cuenta de alojamiento, el sistema crea automáticamente una cuenta FTP para nuestro sitio web que incluye el nombre de usuario y clave requeridos para conectarse al servidor usando este protocolo (si el sistema no configura esta cuenta, podemos hacerlo nosotros mismos desde la opción **FTP Accounts** en el panel de control). Los valores que necesitamos para realizar la conexión son el host (IP o dominio), el usuario y la clave, además del puerto asignado por el servidor para conectarse por medio de FTP (por defecto, 21). Filezilla ofrece dos maneras de insertar esta información: una barra en la parte superior con la que realizar una conexión rápida (Figura 1-11, número 2) y un botón para almacenar múltiples conexiones de uso frecuente (Figura 1-11, número 1).

Figura 1-11: Configuración de conexiones

Si pulsamos el botón para almacenar o acceder a conexiones previas (número 1), Filezilla abre una ventana donde podemos administrar la lista de conexiones disponibles y especificar opciones adicionales de configuración. La ventana incluye botones para crear, renombrar y borrar una conexión (**New Site**, **Rename**, **Delete**), campos donde podemos seleccionar el protocolo que deseamos utilizar (FTP para una conexión normal y SFTP para una conexión segura), el modo de encriptación usado para transferir los archivos y el tipo de cuenta requerida (**Anonymous** para conexiones anónimas o **Normal** para conexiones que requieren usuario y clave). El programa también incluye paneles adicionales para una configuración más avanzada, tal como muestra la Figura 1-12.

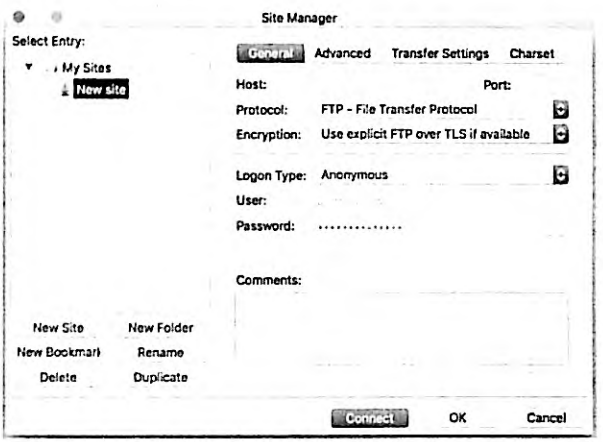

Figura 1-12: Administrador de conexiones

En una situación normal, para establecer la conexión tenemos que insertar el host (el IP o dominio de nuestro sitio web), seleccionar el tipo de cuenta como **Normal**, introducir el nombre de usuario y la contraseña, y dejar el resto de los valores por defecto. Una vez que los ordenadores se conectan, Filezilla muestra la lista de archivos en la pantalla. A la izquierda se

encuentran los archivos en el directorio seleccionado en nuestro ordenador (podemos seleccionar cualquier directorio que queramos en nuestro disco duro), y a la derecha se encuentran los archivos y directorios disponibles en el directorio raíz de nuestra cuenta de alojamiento, como muestra la Figura 1-13.

Figura 1-13: Contenido del directorio raíz

Cuando se crea una cuenta de alojamiento, el sistema incluye varios directorios y archivos para almacenar la información requerida por el servicio (almacenar emails, hacer un seguimiento de la actividad de los usuarios, etc.). El directorio en el que se deben almacenar los archivos de nuestro sitio web se llama *public_html*. Una vez que se abre este directorio, podemos comenzar a subir nuestros archivos arrastrándolos desde el panel de la izquierda al panel de la derecha (ver Figura 1-10).

IMPORTANTE: va más allá del propósito de este libro explicar cómo funciona Filezilla o cualquier otro programa de FTP, pero hemos incluido enlaces a cursos y vídeos en nuestro sitio web con más información. Para acceder a ellos, descargue los contenidos adicionales y haga clic en las opciones **Enlaces** y **Vídeos**.

MAMP

Los documentos HTML se pueden abrir directamente en un ordenador personal. Por ejemplo, si abrimos el archivo index.html con el documento creado en el Listado 1-1, la ventana del navegador muestra el texto HOLA MUNDO!, según ilustra la Figura 1-14 debajo (el resto de los elementos de este documento son estructurales y, por lo tanto, no producen resultados visibles).

Figura 1-14: Documento HTML en el navegador

Lo básico: para abrir un archivo en el navegador, puede seleccionar la opción **Abrir Archivo** desde el menú del navegador o hacer doble clic en el archivo desde el explorador de archivos de Windows (o Finder en ordenadores Apple), y el sistema se encarga de abrir el navegador y cargar el documento.

Aunque la mayoría de los ejemplos de este libro se pueden probar sin subirlos a un servidor, abrir un sitio web completo en un ordenador personal no es siempre posible. Como veremos más adelante, algunos códigos JavaScript solo trabajan cuando se descargan desde un servidor, y tecnologías de servidor como PHP requieren ser alojadas en un servidor para funcionar. Para trabajar con estas clases de documentos existen dos alternativas: podemos obtener una cuenta de alojamiento web de inmediato y usarla para hacer pruebas, o instalar un servidor en nuestro propio ordenador. Esta última opción no hará que se pueda acceder a nuestro sitio web desde Internet, pero nos permite probarlo y experimentar con el código antes de subir la versión final a un servidor real.

Existen varios paquetes que instalan todos los programas necesarios para convertir nuestro ordenador en un servidor. Estos paquetes incluyen un servidor Apache (para despachar archivos web a través del protocolo HTTP), un servidor PHP (para procesar código PHP), y un servidor MySQL (para procesar bases de datos de tipo MySQL que podemos usar para almacenar datos en el servidor). El que recomendamos se llama *MAMP*. Es un paquete gratuito, disponible para ordenadores personales y ordenadores Apple, que podemos descargar desde **www.mamp.info** (la empresa también ofrece una versión comercial avanzada llamada *MAMP PRO*).

MAMP es fácil de instalar y usar. Una vez descargado e instalado, solo necesitamos abrirlo para comenzar a utilizar el servidor.

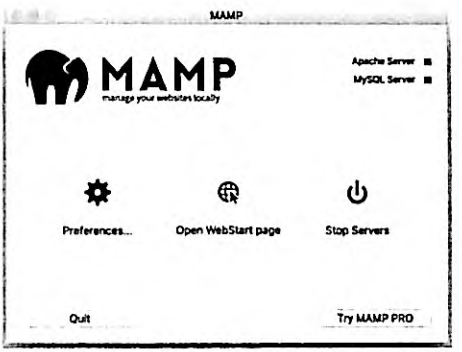

Figura 1-15: Pantalla principal de MAMP

MAMP crea un directorio dentro de su propio directorio llamado *htdocs* donde se supone que debemos almacenar los archivos de nuestro sitio web, pero si lo deseamos, podemos asignar un directorio diferente desde la opción **Preferences**. Esta opción abre una nueva ventana con varias pestañas para su configuración. La pestaña **Web Server** muestra el directorio actual que usa el servidor Apache y ofrece un botón para seleccionar uno diferente, tal como ilustra la Figura 1-16, número 1.

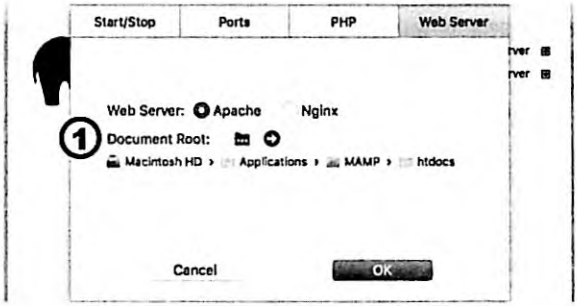

***Figura 1-16:** Directorio del servidor Apache*

Después de seleccionar el directorio en el que se encuentran los archivos de nuestro sitio web, podemos acceder a ellos desde el servidor. Apache crea un dominio especial llamado *localhost* para referenciar al servidor y, por lo tanto, se puede acceder a nuestro sitio web desde la URL http://localhost/. Si queremos acceder a un archivo específico, solo tenemos que agregar el nombre del archivo al final de la URL, tal como hacemos con cualquier otro dominio (por ejemplo, http://localhost/contacto.html).

Hágalo usted mismo: visite **www.mamp.info** y descargue la versión gratuita de MAMP para su sistema (Windows o macOS). Instale el paquete y abra la aplicación. Seleccione la opción **Preferences** (Figura 1-15) y reemplace el directorio htdocs por el directorio que haya creado para su sitio web en el ejemplo anterior (Figura 1-8). Abra el navegador e inserte la URL http://localhost/. Si ha creado el archivo index.html como sugerimos anteriormente, debería ver el texto HOLA MUNDO! en la pantalla (Figura 1-14).

IMPORTANTE: el sistema operativo de Apple incluye su propia versión del servidor Apache, lo que obliga a MAMP a conectar Apache en un puerto diferente para evitar conflictos. Por esta razón, en un ordenador Mac tiene que especificar el puerto 8888 cuando intenta acceder al localhost (http://localhost:8888). Si lo desea, puede cambiar el puerto desde la configuración de MAMP. Haga clic en **Preferences**, seleccione la pestaña **Ports**, y presione el botón **Set Web & MySQL ports**.

Lo básico: la mayoría de los ejemplos de este libro se pueden ejecutar en un ordenador personal sin tener que instalar ningún servidor (le informaremos cuando esto no sea posible), pero si lo desea, puede instalar MAMP para asegurarse de que todo funcione correctamente como lo haría en un servidor real.

Capítulo 2
HTML

2.1 Estructura

A pesar de las innovaciones introducidas por CSS y JavaScript en estos últimos años, la estructura creada por el código HTML sigue siendo la parte fundamental del documento. Esta estructura define el espacio dentro del documento donde el contenido estático y dinámico es posicionado y es la plataforma básica para toda aplicación. Para crear un sitio o una aplicación web, lo primero que debemos hacer es programar el código HTML que define la estructura de cada una de las páginas que lo componen.

IMPORTANTE: los documentos HTML son archivos de texto que se pueden crear con cualquier editor de texto o los editores profesionales indicados en el Capítulo 1. La mayoría de estos editores ofrecen herramientas para ayudarle a escribir sus documentos, pero no controlan la validez del código. Si omite una etiqueta o se olvida de escribir una parte del código, el editor no le advierte sobre el error cometido. Para controlar el código de sus documentos, puede usar herramientas de validación en línea. La más popular para documentos HTML se encuentra disponible en http://validator.w3.org.

Tipo de documento

Debido a que los navegadores son capaces de procesar diferentes tipos de archivos, lo primero que debemos hacer en la construcción de un documento HTML es indicar su tipo. Para asegurarnos de que el contenido de nuestros documentos sea interpretado correctamente como código HTML, debemos agregar la declaración **<!DOCTYPE>** al comienzo del archivo. Esta declaración, similar en formato a las etiquetas HTML, se requiere al comienzo de cada documento para ayudar al navegador a decidir cómo debe generar la página web. Para documentos programados con HTML5, la declaración debe incluir el atributo **html**, según la definimos en el siguiente ejemplo.

```
<!DOCTYPE html>
```

Listado 2-1: Incluyendo la declaración <!DOCTYPE>

Hágalo usted mismo: abra Atom o su editor favorito y cree un nuevo archivo llamado index.html para probar los códigos de este capítulo (también puede usar el archivo creado en el capítulo anterior).

IMPORTANTE: la línea con la declaración **<!DOCTYPE>** debe ser la primera línea de su documento, sin ningún espacio o código previo. Esto activa el modo estándar y obliga a los navegadores a interpretar HTML5 cuando es posible o ignorarlo en caso contrario.

Lo básico: algunos de los elementos y atributos introducidos en HTML5 no están disponibles en viejos navegadores como Internet Explorer. Para saber qué navegadores implementan estos elementos u otras funciones incorporadas por HTML5, visite www.caniuse.com. Este sitio web ofrece una lista de todos los elementos, atributos, propiedades CSS, y códigos JavaScript disponibles en HTML5, junto con los navegadores que los admiten.

Elementos estructurales

Como mencionamos en el Capítulo 1, los elementos HTML conforman una estructura de tipo árbol con el elemento **<html>** como su raíz. Esta estructura presenta múltiples niveles de organización, con algunos elementos a cargo de definir secciones generales del documento y otros encargados de representar secciones menores o contenido. Los siguientes son los elementos disponibles para definir la columna vertebral de la estructura y facilitar la información que el navegador necesita para mostrar la página en la pantalla.

<html>—Este elemento delimita el código HTML. Puede incluir el atributo **lang** para definir el idioma del contenido del documento.

<head>—Este elemento se usa para definir la información necesaria para configurar la página web, como el título, el tipo de codificación de caracteres y los archivos externos requeridos por el documento.

<body>—Este elemento delimita el contenido del documento (la parte visible de la página).

Después de declarar el tipo de documento, tenemos que construir la estructura de tipo árbol con los elementos HTML, comenzando por el elemento **<html>**. Este elemento puede incluir el atributo **lang** para declarar el idioma en el que vamos a escribir el contenido de la página, tal como muestra el siguiente ejemplo.

```
<!DOCTYPE html>
<html lang="es">

</html>
```

***Listado 2-2:** Incluyendo el elemento* `<html>`

Lo básico: existen varios valores disponibles para el atributo **lang**, incluidos **en** para inglés, **es** para español, **fr** para francés, entre otros. Para obtener una lista completa, visite https://en.wikipedia.org/wiki/List_of_ISO_639-1_codes.

El código HTML insertado entre las etiquetas **<html>** se tiene que dividir en dos secciones principales: la cabecera y el cuerpo. Por supuesto, la cabecera va primero y, al igual que el resto de los elementos estructurales, está compuesta por etiquetas de apertura y cierre.

```
<!DOCTYPE html>
<html lang="es">
<head>

</head>
</html>
```

Listado 2-3: Incluyendo el elemento <head>

Entre las etiquetas **<head>** debemos definir el título de la página web, declarar el tipo de codificación de caracteres, facilitar información general acerca del documento, e incorporar los archivos externos con estilos y códigos necesarios para generar la página. Excepto por el título e iconos, el resto de la información insertada en medio de estas etiquetas no es visible para el usuario.

La otra sección que forma parte de la organización principal de un documento HTML es el cuerpo, la parte visible del documento que se especifica con el elemento **<body>**.

```
<!DOCTYPE html>
<html lang="es">
<head>

</head>
<body>

</body>
</html>
```

Listado 2-4: Incluyendo el elemento <body>

Lo básico: como ya mencionamos, la estructura HTML puede describirse como un árbol, con el elemento **<html>** como su raíz, pero otra forma de definir la relación entre los elementos es describirlos como padres, hijos o hermanos, de acuerdo a sus posiciones en la estructura. Por ejemplo, en un documento HTML típico, el elemento **<body>** es hijo del elemento **<html>** y hermano del elemento **<head>**. Ambos, **<body>** y **<head>**, tienen al elemento **<html>** como su padre.

La estructura básica ya está lista. Ahora tenemos que construir la página, comenzando por la definición de la cabecera. La cabecera incluye toda la información y los recursos necesarios para generar la página. Los siguientes son los elementos disponibles para este propósito.

<title>—Este elemento define el título de la página.

<base>—Este elemento define la URL usada por el navegador para establecer la ubicación real de las URL relativas. El elemento debe incluir el atributo **href** para declarar la URL base. Cuando se declara este elemento, en lugar de la URL actual, el navegador usa la URL asignada al atributo **href** para completar las URL relativas.

<meta>—Este elemento representa metadatos asociados con el documento, como la descripción del documento, palabras claves, el tipo de codificación de caracteres, etc. El elemento puede incluir los atributos **name** para describir el tipo de metadata, **content**

para especificar el valor, y `charset` para declarar el tipo de codificación de caracteres a utilizar para procesar el contenido.

<link>—Este elemento especifica la relación entre el documento y un recurso externo (generalmente usado para cargar archivos CSS). El elemento puede incluir los atributos `href` para declarar la ubicación del recurso, `rel` para definir el tipo de relación, `media` para especificar el medio al que el recurso está asociado (pantalla, impresora, etc.), y `type` y `sizes` para declarar el tipo de recurso y su tamaño (usado a menudo para cargar iconos).

<style>—Este elemento se usa para declarar estilos CSS dentro del documento (estudiado en el Capítulo 3).

<script>—Este elemento se usa para cargar o declarar código JavaScript (estudiado en el Capítulo 6).

Lo primero que tenemos que hacer cuando declaramos la cabecera del documento es especificar el título de la página con el elemento `<title>`. Este texto es el que muestran los navegadores en la parte superior de la ventana, y es lo que los usuarios ven cuando buscan información en nuestro sitio web por medio de motores de búsqueda como Google o Yahoo.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Este texto es el título del documento</title>
</head>
<body>

</body>
</html>
```

***Listado 2-5:** Incluyendo el elemento* `<title>`

Hágalo usted mismo: reemplace el código en su archivo index.html por el código del Listado 2-5 y abra el documento en su navegador (para abrirlo, puede hacer doble clic en el archivo o seleccionar la opción **Abrir Archivo** desde el menú **Archivos** en su navegador). Debería ver el texto especificado entre las etiquetas `<title>` en la parte superior de la ventana.

Lo básico: el elemento `<title>` en el ejemplo del Listado 2-5 se ha desplazado hacia la derecha. El espacio en blanco en el lado izquierdo se usa para ayudar al desarrollador a visualizar la posición del elemento dentro de la jerarquía del documento Este espacio se genera automáticamente por editores como Atom, pero puede hacerlo usted mismo cuando lo necesite pulsando la tecla Tab (Tabulador) en su teclado (los navegadores ignoran los espacios en blanco y los saltos de líneas que se encuentran fuera de los elementos).

Además del título, también tenemos que declarar los metadatos del documento. Los metadatos incluyen información acerca de la página que los navegadores, y también los motores de búsqueda, utilizan para generar y clasificar la página web. Los valores se declaran con el elemento `<meta>`. Este elemento incluye varios atributos, pero cuáles usemos

dependerá del tipo de información que queremos declarar. Por ejemplo, el valor más importante es el que define la tabla de caracteres a utilizar para presentar el texto en pantalla, el cual se declara con el atributo **charset**.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Este texto es el título del documento</title>
  <meta charset="utf-8">
</head>
<body>

</body>
</html>
```

Listado 2-6: Incluyendo el elemento <meta>

Lo básico: el ejemplo del Listado 2-6 define el grupo de caracteres como **utf-8**, que es el que se recomienda debido a que incluye todos los caracteres utilizados en la mayoría de los idiomas, pero existen otros disponibles. Para más información, visite nuestro sitio web y siga los enlaces de este capítulo.

Se pueden incluir múltiples elementos **<meta>** para declarar información adicional. Por ejemplo, dos datos que los navegadores pueden considerar a la hora de procesar nuestros documentos son la descripción de la página y las palabras claves que identifican su contenido. Estos elementos **<meta>** requieren el atributo **name** con los valores "description" y "keywords", y el atributo **content** con el texto que queremos asignar como descripción y palabras clave (las palabras clave se deben separar por comas).

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Este texto es el título del documento</title>
  <meta charset="utf-8">
  <meta name="description" content="Este es un documento HTML5">
  <meta name="keywords" content="HTML, CSS, JavaScript">
</head>
<body>

</body>
</html>
```

Listado 2-7: Agregando información adicional con el elemento <meta>

Otro elemento importante de la cabecera del documento es **<link>**. Este elemento se usa para incorporar al documento estilos, códigos, imágenes o iconos desde archivos externos. Por ejemplo, algunos navegadores muestran un icono en la parte superior de la ventana junto con el título de la página. Para cargar este icono, tenemos que incluir un elemento **<link>** con el atributo **rel** definido como **icon**, el atributo **href** con la ubicación del archivo que contiene

el icono, el atributo **type** para especificar el formato con el que se ha creado el icono, y el atributo **sizes** con el ancho y la altura del icono separados por la letra x.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Este texto es el título del documento</title>
  <meta charset="utf-8">
  <meta name="description" content="Este es un documento HTML5">
  <meta name="keywords" content="HTML, CSS, JavaScript">
  <link rel="icon" href="imagenes/favicon.png" type="image/png"
sizes="16x16">
</head>
<body>

</body>
</html>
```

***Listado 2-8:** Incluyendo el icono del documento*

El navegador tiene poco espacio para mostrar el icono, por lo tanto el tamaño típico de esta imagen es de unos 16 píxeles por 16 píxeles. La Figura 2-1 muestra cómo se ve la ventana cuando abrimos un documento que contiene un icono (en este caso, se muestra una imagen con la letra M en el lado izquierdo del título).

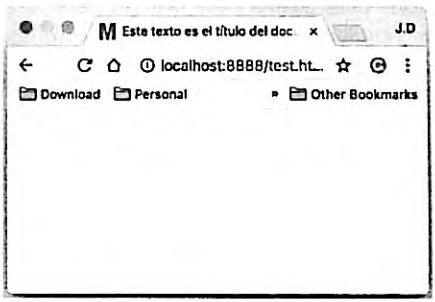

***Figura 2-1:** El icono del documento en el navegador*

Hágalo usted mismo: actualice el código en su archivo index.html con el ejemplo del Listado 2-8. El icono se carga desde el archivo favicon.png que debe copiar dentro del directorio de su proyecto. Puede descargar este archivo desde nuestro sitio web o usar el suyo.

Lo básico: el valor asignado al atributo **type** del elemento **<link>** debe ser especificado como un tipo MIME. Todo archivo tiene un tipo MIME asociado para indicar al sistema el formato de su contenido. Por ejemplo, el tipo MIME de un archivo HTML es **text/html**. Existe un tipo MIME para cada tipo de archivo disponible, que incluye **image/jpeg** e **image/png** para imágenes JPEG y PNG. Para obtener una lista completa, visite nuestro sitio web y siga los enlaces de este capítulo.

El elemento **<link>** se usa comúnmente para cargar archivos CSS con los estilos necesarios para generar la página web. Por ejemplo, el siguiente documento carga el archivo misestilos.css. Después de cargar el archivo, todos los estilos declarados en su interior se aplican a los elementos del documento. En este caso, solo necesitamos incluir el atributo **rel** para declarar el tipo de recurso (para hojas de estilo CSS debemos asignar el valor "stylesheet") y el atributo **href** con la URL que determina la ubicación del archivo (estudiaremos cómo crear esta clase de archivos y definir estilos CSS en el Capítulo 3).

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Este texto es el título del documento</title>
  <meta charset="utf-8">
  <meta name="description" content="Este es un documento HTML5">
  <meta name="keywords" content="HTML, CSS, JavaScript">
  <link rel="stylesheet" href="misestilos.css">
</head>
<body>

</body>
</html>
```

Listado 2-9: Cargando un archivo CSS con el elemento **<link>**

Con la cabecera lista, es hora de construir el cuerpo. Esta estructura (el código entre las etiquetas **<body>**) es la encargada de generar la parte visible de nuestro documento (la página web).

HTML siempre ha ofrecido diferentes maneras de construir y organizar la información en el cuerpo del documento. Uno de los primeros elementos utilizados con este propósito fue **<table>** (tabla). Este elemento permitía a los desarrolladores organizar datos, textos, imágenes, así como herramientas en filas y columnas de celdas. Con la introducción de CSS, la estructura generada por estas tablas ya no resultaba práctica, por lo que los desarrolladores comenzaron a implementar un elemento más flexible llamado ** ** (división). Pero ** **, así como **<table>**, no facilita demasiada información acerca de las partes del cuerpo que representa. Cualquier cosa, desde imágenes hasta menús, texto, enlaces, códigos o formularios, se puede insertar entre las etiquetas de apertura y cierre de un elemento ** **. En otras palabras, el nombre **div** solo especifica una división en el cuerpo, como una celda en una tabla, pero no ofrece ninguna pista acerca del tipo de división que está creando, cuál es su propósito o qué contiene. Esta es la razón por la que HTML5 introdujo nuevos elementos con nombres más descriptivos que permiten a los desarrolladores identificar cada parte del documento. Estos elementos no solo ayudan a los desarrolladores a crear el documento, sino que además informan al navegador sobre el propósito de cada sección. La siguiente lista incluye todos los elementos disponibles para definir la estructura del cuerpo.

** **—Este elemento define una división genérica. Se usa cuando no se puede aplicar ningún otro elemento.

<main>—Este elemento define una división que contiene el contenido principal del documento (el contenido que representa el tema central de la página).

<nav>—Este elemento define una división que contiene ayuda para la navegación, como el menú principal de la página o bloques de enlaces necesarios para navegar en el sitio web.

<section>—Este elemento define una sección genérica. Se usa frecuentemente para separar contenido temático, o para generar columnas o bloques que ayudan a organizar el contenido principal.

<aside>—Este elemento define una división que contiene información relacionada con el contenido principal pero que no es parte del mismo, como referencias a artículos o enlaces que apuntan a publicaciones anteriores.

<article>—Este elemento representa un artículo independiente, como un mensaje de foro, el artículo de una revista, una entrada de un blog, un comentario, etc.

<header>—Este elemento define la cabecera del cuerpo o de secciones dentro del cuerpo.

<footer>—Este elemento define el pie del cuerpo o de secciones dentro del cuerpo.

Estos elementos han sido definidos con el propósito de representar secciones específicas de una página web. Aunque son flexibles y se pueden implementar en diferentes partes del diseño, todos siguen un patrón que se encuentra comúnmente en la mayoría de los sitios web. La Figura 2-2, a continuación, ilustra este tipo de diseño.

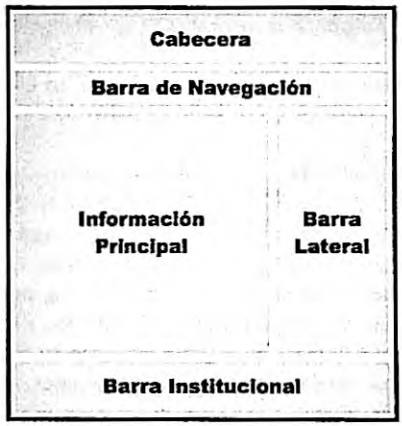

Figura 2-2: Representación visual de un diseño web tradicional

A pesar de que cada desarrollador crea sus propios diseños, en general podremos describir todo sitio web considerando estas secciones. En la barra superior, descrita como **cabecera** en la Figura 2-2, ubicamos el logo, el nombre del sitio, los subtítulos y una descripción breve de nuestro sitio o página web. En la **barra de navegación** situada debajo es donde la mayoría de los desarrolladores ofrecen un menú o una lista de enlaces para navegar en el sitio. El contenido relevante de la página se ubica en el medio del diseño, donde generalmente encontramos artículos o noticias, y también enlaces a documentos relacionados o recursos. En el ejemplo de la Figura 2-2, esta sección se ha dividido en dos columnas, **información principal** y **barra lateral**, pero los diseñadores la adaptan a sus necesidades insertando columnas adicionales o dividiendo las columnas en bloques más pequeños. En la parte inferior de un diseño tradicional, nos encontramos con otra barra llamada **barra institucional**. La llamamos de este modo porque en este área es

donde mostramos información general acerca del sitio web, el autor, la compañía, los enlaces relacionados con reglas de uso, términos y condiciones, el mapa del sitio, etc.

Como mencionamos anteriormente, los elementos de HTML5 se han diseñado siguiendo este patrón. En la Figura 2-3 aplicamos los elementos introducidos anteriormente para definir el diseño de la Figura 2-2.

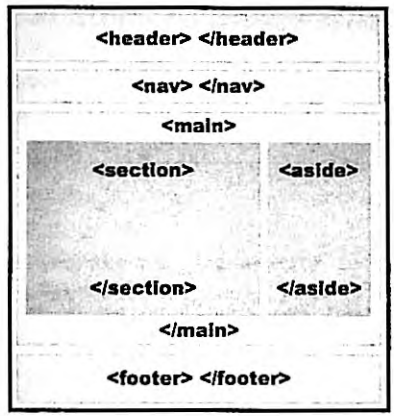

Figura 2-3: Representación de la estructura de un documento usando elementos de HTML5

Los elementos se declaran en el documento en el mismo orden en el que se presentarán en pantalla, desde la parte superior a la inferior y de izquierda a derecha (este orden se puede modificar por medio de estilos CSS, como veremos en el Capítulo 4). El primer elemento de un diseño tradicional es **<header>**. No debemos confundir este elemento con el elemento **<head>** utilizado anteriormente para crear la cabecera del documento. Al igual que **<head>**, el elemento **<header>** se ha definido para facilitar información introductoria, como títulos o subtítulos, pero no para el documento, sino para el cuerpo o secciones dentro del cuerpo del documento. En el siguiente ejemplo, este elemento se usa para definir el título de la página web.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Este texto es el título del documento</title>
  <meta charset="utf-8">
  <meta name="description" content="Este es un documento HTML5">
  <meta name="keywords" content="HTML, CSS, JavaScript">
  <link rel="stylesheet" href="misestilos.css">
</head>
<body>
  <header>
    Este es el título
  </header>
</body>
</html>
```

Listado 2-10: Incluyendo el elemento **<header>**

La inserción del elemento **<header>** representa el comienzo del cuerpo y de la parte visible del documento. De ahora en adelante, podremos ver el resultado de la ejecución del código en la ventana del navegador.

Hágalo usted mismo: reemplace el código en su archivo index.html por el código del Listado 2-10 y abra el documento en su navegador. Debería ver el título de la página en la pantalla.

La siguiente sección de nuestro ejemplo es la **barra de navegación**. Esta barra define una sección con ayuda para la navegación y se representa con el elemento **<nav>**.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Este texto es el título del documento</title>
  <meta charset="utf-8">
  <meta name="description" content="Este es un documento HTML5">
  <meta name="keywords" content="HTML, CSS, JavaScript">
  <link rel="stylesheet" href="misestilos.css">
</head>
<body>
  <header>
    Este es el título
  </header>
  <nav>
    Principal | Fotos | Videos | Contacto
  </nav>
</body>
</html>
```

***Listado 2-11:** Incluyendo el elemento* **<nav>**

La estructura y el orden que decidimos implementar depende de lo que nuestro sitio web o aplicación requieran. Los elementos HTML son bastante flexibles y solo nos dan ciertos parámetros con los que trabajar, pero el modo en que los usemos depende de nosotros. Un ejemplo de esta versatilidad es que el elemento **<nav>** se podría insertar dentro de etiquetas **<header>** o en otra sección del cuerpo. Sin embargo, siempre debemos considerar que estos elementos se han creado para ofrecer información adicional al navegador, y ayudar a cada nuevo programa y dispositivo a identificar las partes relevantes del documento. Si queremos mantener nuestro código HTML portable y legible, es mejor seguir los estándares establecidos por estos elementos. El elemento **<nav>** se ha creado con la intención de contener ayuda para la navegación, como el menú principal o bloques de enlaces importantes, y deberíamos usarlo con este propósito.

Otro ejemplo de especificidad es el que ofrecen los elementos **<main>**, **<section>**, y **<aside>**, que se han diseñado para organizar el contenido principal del documento. En nuestro diseño, estos elementos representan las secciones que llamamos **Información principal** y **Barra lateral**. Debido a que la sección **Información principal** abarca más, su contenido generalmente se representa por elementos **<section>** (uno o varios, dependiendo del diseño), y debido al tipo de información que contiene, el elemento **<aside>** se ubica en los laterales de la página. La mayoría del tiempo, estos dos elementos son suficientes para

representar el contenido principal, pero como se pueden usar en otras áreas del documento, se implementa el elemento **<main>** para agruparlos, como lo muestra el siguiente ejemplo.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Este texto es el título del documento</title>
  <meta charset="utf-8">
  <meta name="description" content="Este es un documento HTML5">
  <meta name="keywords" content="HTML, CSS, JavaScript">
  <link rel="stylesheet" href="misestilos.css">
</head>
<body>
  <header>
    Este es el título
  </header>
  <nav>
    Principal | Fotos | Videos | Contacto
  </nav>
  <main>
    <section>
      Artículos
    </section>
    <aside>
      Cita del artículo uno
      Cita del artículo dos
    </aside>
  </main>
</body>
</html>
```

Listado 2-12: Organizando el contenido principal

El elemento **<aside>** describe la información que contiene, no un lugar en la estructura, por lo que se podría ubicar en cualquier parte del diseño, y se puede usar mientras su contenido no se considere el contenido principal del documento.

IMPORTANTE: los elementos que representan cada sección del documento se listan en el código uno encima del otro, pero en la página web algunas de estas secciones se muestran una al lado de la otra (por ejemplo, las columnas creadas por las secciones **Información principal** y **Barra lateral** se muestran en una misma línea en la página web). Si abre el documento del Listado 2-12 en su navegador, verá que los textos se muestran uno por línea. Esto se debe a que HTML5 delega la tarea de presentar el documento a CSS. Para mostrar las secciones en el lugar correcto, debemos asignar estilos CSS a cada elemento del documento. Estudiaremos CSS en los Capítulos 3 y 4.

El diseño considerado anteriormente (Figura 2-2) es el más común de todos y representa la estructura básica de la mayoría de los sitios web que encontramos hoy en día, pero es también una muestra de cómo se muestra el contenido importante de una página web en pantalla.

Como los artículos de un diario, las páginas web generalmente presentan la información dividida en secciones que comparten características similares. El elemento **<article>** nos permite identificar cada una de estas partes. En el siguiente ejemplo, implementamos este elemento para representar las publicaciones que queremos mostrar en la sección principal de nuestra página web.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Este texto es el título del documento</title>
  <meta charset="utf-8">
  <meta name="description" content="Este es un documento HTML5">
  <meta name="keywords" content="HTML, CSS, JavaScript">
  <link rel="stylesheet" href="misestilos.css">
</head>
<body>
  <header>
    Este es el título
  </header>
  <nav>
    Principal | Fotos | Videos | Contacto
  </nav>
  <main>
    <section>
      <article>
        Este es el texto de mi primer artículo
      </article>
      <article>
        Este es el texto de mi segundo artículo
      </article>
    </section>
    <aside>
      Cita del artículo uno
      Cita del artículo dos
    </aside>
  </main>
</body>
</html>
```

***Listado 2-13:** Incluyendo el elemento* `<article>`

En este punto, ya contamos con la cabecera y el cuerpo del documento, secciones con ayuda para la navegación y el contenido, e información adicional a un lado de la página. Lo único que nos queda por hacer es cerrar el diseño y finalizar el cuerpo del documento. Con este fin, HTML ofrece el elemento **<footer>**.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Este texto es el titulo del documento</title>
  <meta charset="utf-8">
  <meta name="description" content="Este es un documento HTML5">
```

```
  <meta name="keywords" content="HTML, CSS, JavaScript">
  <link rel="stylesheet" href="misestilos.css">
</head>
<body>
  <header>
    Este es el título
  </header>
  <nav>
    Principal | Fotos | Videos | Contacto
  </nav>
  <main>
    <section>
      <article>
        Este es el texto de mi primer artículo
      </article>
      <article>
        Este es el texto de mi segundo artículo
      </article>
    </section>
    <aside>
      Cita del artículo uno
      Cita del artículo dos
    </aside>
  </main>
  <footer>
    &copy; Derechos Reservados 2016
  </footer>
</body>
</html>
```

Listado 2-14: Incluyendo el elemento `<footer>`

En un diseño web tradicional (Figura 2-2), la sección **Barra institucional** se define con el elemento **`<footer>`**. Esto se debe a que la sección representa el final (o pie) de nuestro documento y se usa comúnmente para compartir información general acerca del autor del sitio o la compañía detrás del proyecto, como derechos de autor, términos y condiciones, etc.

El elemento **`<footer>`** se usa para representar el final del documento y tiene el objetivo principal ya mencionado, sin embargo, este elemento y el elemento **`<header>`** también se pueden utilizar dentro del cuerpo para representar el comienzo y final de una sección.

Lo básico: el ejemplo del Listado 2-14 incluye la cadena de caracteres **`&copy;`** al pie del documento. Sin embargo, cuando se carga el documento, el navegador reemplaza estos caracteres por el carácter de derechos de autor (©). Estas cadenas de caracteres se denominan *entidades* (*character entities*) y representan caracteres especiales que no se encuentran en el teclado o tienen un significado especial en HTML, como el carácter de derechos de autor (©), el de marca registrada (®) o los paréntesis angulares usados por HTML para definir los elementos (< y >). Cuando necesite incluir en sus textos uno de estos caracteres, debe escribir la entidad en su lugar. Por ejemplo, si quiere incluir los caracteres < y > dentro de un texto, debe representarlos con las cadenas de caracteres **`&lt;`** y **`&gt;`**. Otras entidades de uso común son **`&amp;`** (&), **`&quot;`** ("), **`&apos;`** ('), **`&pound;`** (£), y

`&euro;` (€). Para obtener una lista completa, visite nuestro sitio web y siga los enlaces de este capítulo.

Atributos globales

Aunque la mayoría de los elementos estructurales tienen un propósito implícito que se refleja en sus nombres, esto no significa que se deban usar solo una vez en el mismo documento. Por ejemplo, algunos elementos como `<section>` y `<aside>` se pueden utilizar muchas veces para representar diferentes partes de la estructura, y otros como `<div>` aún son implementados de forma repetida para separar contenido dentro de secciones. Por esta razón, HTML define atributos globales que podemos usar para asignar identificadores personalizados a cada elemento.

id—Este atributo nos permite asignar un identificador único a un elemento.

class—Este atributo asigna el mismo identificador a un grupo de elementos.

El atributo `id` identifica elementos independientes con un valor único, mientras que el valor del atributo `class` se puede duplicar para asociar elementos con características similares. Por ejemplo, si tenemos dos o más elementos `<section>` que necesitamos diferenciar entre sí, podemos asignar el atributo `id` a cada uno con valores que declaran sus propósitos.

```
<main>
  <section id="noticias">
    Artículos largos
  </section>
  <section id="noticiaslocales">
    Artículos cortos
  </section>
  <aside>
    Quote from article one
    Quote from article two
  </aside>
</main>
```

Listado 2-15: Identificando elementos con el atributo `id`

El ejemplo del Listado 2-15 incluye dos elementos `<section>` en la sección principal del documento para separar artículos de acuerdo a su extensión. Debido a que el contenido de estos elementos es diferente, requieren distintos estilos y, por lo tanto, tenemos que identificarlos con diferentes valores. El primer elemento `<section>` se ha identificado con el valor "noticias" y el segundo elemento con el valor "noticiaslocales".

Por otro lado, si lo que necesitamos es identificar un grupo de elementos con características similares, podemos usar el atributo `class`. El siguiente ejemplo divide el contenido de una sección con elementos `<div>`. Debido a que todos tienen un contenido similar, compartirán los mismos estilos y, por lo tanto, deberíamos identificarlos con el mismo valor (todo son de la misma clase).

```
<main>
  <section>
    <div class="libros">Libro: IT, Stephen King</div>
    <div class="libros">Libro: Carrie, Stephen King</div>
    <div class="libros">Libro: El resplandor, Stephen King</div>
    <div class="libros">Libro: Misery, Stephen King</div>
  </section>
  <aside>
    Cita del artículo uno
    Cita del artículo dos
  </aside>
</main>
```

Listado 2-16: Identificando elementos con el atributo `class`

En el código del Listado 2-16, tenemos un único elemento `<section>` con el que representamos el contenido principal del documento, pero hemos creado varias divisiones con elementos `<div>` para organizar el contenido. Debido a que estos elementos se han identificado con el atributo `class` y el valor "libros", cada vez que accedemos o modificamos elementos referenciando la clase `libros`, todos estos elementos se ven afectados.

IMPORTANTE: los valores de los atributos `id` y `class` son usados por algunos elementos HTML para identificar otros elementos y también por reglas CSS y código JavaScript para acceder y modificar elementos específicos en el documento. En próximos capítulos veremos algunos ejemplos prácticos.

Lo básico: los valores asignados a estos atributos son arbitrarios. Puede asignarles cualquier valor que desee, siempre y cuando no incluya ningún espacio en blanco (el atributo `class` utiliza espacios para asignar múltiples clases a un mismo elemento). Así mismo, para mantener su sitio web compatible con todos los navegadores, debería usar solo letras y números, siempre comenzar con una letra y evitar caracteres especiales.

2.2 Contenido

Hemos concluido la estructura básica de nuestro sitio web, pero todavía tenemos que trabajar en el contenido. Los elementos HTML estudiados hasta el momento nos ayudan a identificar cada sección del diseño y asignarles un propósito, pero lo que realmente importa en una página web es lo que hay dentro de esas secciones. Debido a que esta información está compuesta por diferentes elementos visuales, como títulos, textos, imágenes y vídeos, entre otros, HTML define varios elementos para representarla.

Texto

El medio más importante que puede incluir un documento es texto. HTML define varios elementos para determinar el propósito de cada palabra, frase, o párrafo en el documento. El siguiente elemento se usa para representar títulos.

<h1>—Este elemento representa un título. El título se declara entre las etiquetas de apertura y cierre. HTML también incluye elementos adicionales para representar subtítulos, hasta seis niveles (**<h2>**, **<h3>**, **<h4>**, **<h5>**, y **<h6>**).

Cada vez que queremos insertar un título o un subtítulo en el documento, tenemos que incluirlo dentro de etiquetas **<h>**. Por ejemplo, el documento que hemos creado en la sección anterior incluye el título de la página. Como este es el título principal, debería representarse con el elemento **<h1>**, tal como ilustra el siguiente ejemplo.

```
<header>
  <h1>Este es el título</h1>
</header>
```

***Listado 2-17:** Incluyendo el elemento* `<h1>`

Los navegadores otorgan estilos por defecto a los elementos **<h>** que incluyen márgenes y diferentes tamaños de letras, dependiendo de la jerarquía (**<h1>** es el de más alta jerarquía y **<h6>** el de menor jerarquía). La Figura 2-4, debajo, muestra cómo se ve el texto dentro de un elemento **<h1>** con los estilos por defecto.

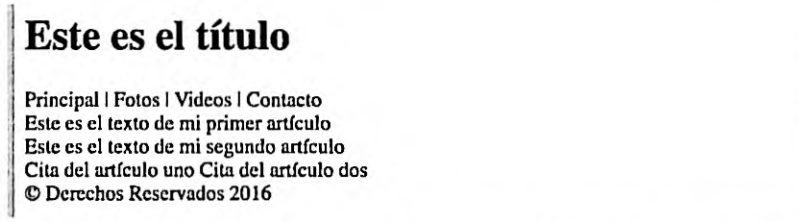

***Figura 2-4:** Título representado por un elemento* `<h1>` *con estilos por defecto*

Hágalo usted mismo: reemplace el elemento **<header>** en su archivo index.html por el código del Listado 2-17. Abra el documento en su navegador. Debería ver algo similar a lo que se muestra en la Figura 2-4.

Los siguientes son los elementos que ofrece HTML para representar el cuerpo del texto.

<p>—Este elemento representa un párrafo. Por defecto, los navegadores le asignan un margen en la parte superior para separar un párrafo de otro.

<pre>—Este elemento representa un texto con formato predefinido, como código de programación o un poema que requiere que los espacios asignados a cada carácter y los saltos de línea se muestren como se han declarado originalmente.

****—Este elemento puede contener un párrafo, una frase o una palabra. No aplica ningún estilo al texto pero se usa para asignar estilos personalizados, como veremos en próximos capítulos.

El elemento **<p>** se utiliza ampliamente para representar el cuerpo del texto. Por defecto, los navegadores les asignan estilos que incluyen márgenes y un salto de línea para diferenciar

un párrafo de otro. Debido a estas características, también podemos utilizar los elementos **<p>** para dar formato a líneas de texto, como las citas de nuestro ejemplo.

```
<aside>
  <p>Cita del artículo uno</p>
  <p>Cita del artículo dos</p>
</aside>
```

Listado 2-18: Definiendo líneas de texto con el elemento **<p>**

El Listado 2-18 presenta las citas dentro del elemento **<aside>** de ejemplos anteriores con elementos **<p>**. Ahora, el navegador muestra cada cita en una línea distinta.

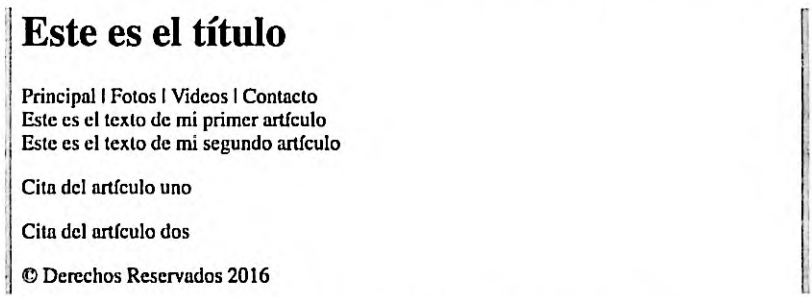

Figura 2-5: Líneas de texto definidas con elementos **<p>**

Cuando un párrafo incluye múltiples espacios, el elemento **<p>** automáticamente reduce ese espacio a solo un carácter e ignora el resto. El elemento también hace algo similar con los saltos de línea. Todo salto de línea introducido en el documento no se considera cuando el texto se muestra en la pantalla. Si queremos que estos espacios y saltos de línea se muestren al usuario, en lugar de usar el elemento **<p>** tenemos que usar el elemento **<pre>**.

```
<article>
  <pre>
    La muerte es una quimera: porque mientras yo existo, no existe la
muerte;
    y cuando existe la muerte, ya no existo yo.
    Epicuro de Samos
  </pre>
</article>
```

Listado 2-19: Mostrando texto en su formato original

El ejemplo del Listado 2-19 define un elemento **<article>** que contiene una cita de Epicuro de Samos. Como usamos el elemento **<pre>**, los saltos de línea son considerados por el navegador y las frases se muestran una por línea, tal como se han definido en el código.

```
La muerte es una quimera: porque mientras yo existo, no existe la muerte;
y cuando existe la muerte, ya no existo yo.
Epicuro de Samos
```

Figura 2-6: Texto introducido con un elemento `<pre>`

Hágalo usted mismo: reemplace el elemento `<article>` en su archivo index.html por el código del Listado 2-19. Abra el documento en su navegador. Debería ver algo similar a lo que se muestra en la Figura 2-6.

El elemento `<pre>` se configura por defecto con márgenes y un tipo de letra que respeta el formato asignado al texto original, lo que lo hace apropiado para presentar código de programación y cualquier clase de texto con formato predefinido. En casos como el del ejemplo anterior, donde lo único que necesitamos es incluir saltos de línea dentro del párrafo, podemos usar otros elementos que se han diseñado específicamente con este propósito.

`<br>`—Este elemento se usa para insertar saltos de línea.

`<wbr>`—Este elemento sugiere la posibilidad de un salto de línea para ayudar al navegador a decidir dónde cortar el texto cuando no hay suficiente espacio para mostrarlo entero.

Estos elementos se insertan dentro del texto para generar saltos de línea. Por ejemplo, podemos escribir el párrafo anterior en una sola línea e insertar elementos `<br>` al final de cada frase para presentarlas en líneas aparte.

```
<article>
  <p>La muerte es una quimera: porque mientras yo existo, no existe la
muerte;<br>y cuando existe la muerte, ya no existo yo.<br>Epicuro de
Samos</p>
</article>
```

***Listado 2-20:** Agregando saltos de línea a un párrafo con el elemento* `<br>`

A diferencia de los elementos `<p>` y `<pre>`, los elementos `<br>` y `<wbr>` no asignan ningún margen o tipo de letra al texto, por lo que las líneas se muestran como si pertenecieran al mismo párrafo y con el tipo de letra definida por defecto.

La muerte es una quimera: porque mientras yo existo, no existe la muerte;
y cuando existe la muerte, ya no existo yo.
Epicuro de Samos

Figura 2-7: Saltos de línea generadas por elementos `<br>`

Debido a que no todas las palabras en un texto tienen el mismo énfasis, HTML incluye los siguientes elementos para declarar un significado especial a palabras individuales o frases completas.

****—Este elemento se usa para indicar énfasis. El texto se muestra por defecto con letra cursiva.

****—Este elemento es utiliza para indicar importancia. El texto se muestra por defecto en negrita.

<i>—Este elemento representa una voz alternativa o un estado de humor, como un pensamiento, un término técnico, etc. El texto se muestra por defecto con letra cursiva.

<u>—Este elemento representa texto no articulado. Por defecto se muestra subrayado.

<b>—Este elemento se usa para indicar importancia. Debería ser implementado solo cuando ningún otro elemento es apropiado para la situación. El texto se muestra por defecto en negrita.

Estos elementos se pueden utilizar para resaltar títulos o etiquetas, o para destacar palabras o frases en un párrafo, según muestra el siguiente ejemplo.

```
<article>
  <p>La muerte es una <em>quimera</em>: porque mientras yo existo, no
existe la <i>muerte</i>;<br>y cuando existe la <i>muerte</i>,
<strong>ya no existo yo</strong>.<br>Epicuro de Samos</p>
</article>
```

Listado 2-21: Resaltando texto

A menos que especifiquemos diferentes estilos con CSS, el texto dentro de estos elementos se muestra con los estilos por defecto, como ilustra la Figura 2-8.

La muerte es una *quimera*: porque mientras yo existo, no existe la *muerte*;
y cuando existe la *muerte*, ya no existo yo.
Epicuro de Samos

Figura 2-8: Texto resaltado

La especificidad de elementos estructurales también se manifiesta en algunos de los elementos utilizados para definir el contenido. Por ejemplo, HTML incluye los siguientes elementos para insertar textos que tienen un propósito claramente definido.

<mark>—Este elemento resalta texto que es relevante en las circunstancias actuales (por ejemplo, términos que busca el usuario).

<small>—Este elemento representa letra pequeña, como declaraciones legales, descargos, etc.

<cite>—Este elemento representa el autor o título de una obra, como un libro, una película, etc.

<address>—Este elemento representa información de contacto. Se implementa con frecuencia dentro de los pies de página para definir la dirección de la empresa o el sitio web.

<time>—Este elemento representa una fecha en formato legible para el usuario. Incluye el atributo **datetime** para especificar un valor en formato de ordenador y el atributo **pubdate**, el cual indica que el valor asignado al atributo **datetime** representa la fecha de publicación.

<code>—Este elemento representa código de programación. Se usa en conjunto con el elemento **<pre>** para presentar código de programación en el formato original.

<data>—Este elemento representa datos genéricos. Puede incluir el atributo **value** para especificar el valor en formato de ordenador (por ejemplo, **<data value="32">Treinta y Dos</data>**).

Como estos elementos representan información específica, normalmente se utilizan para complementar el contenido de otros elementos. Por ejemplo, podemos usar el elemento **<time>** para declarar la fecha en la que un artículo se ha publicado y otros elementos como **<mark>** y **<cite>** para otorgarle significado a algunas partes del texto.

```
<article>
  <header>
    <h1>Título del artículo</h1>
    <time datetime="2016-10-12" pubdate>publicado 12-10-2016</time>
  </header>
  <p>La muerte es una quimera: porque mientras yo <mark>existo</mark>,
no existe la muerte;<br>y cuando existe la muerte, ya no existo
yo.<br><cite>Epicuro de Samos</cite></p>
</article>
```

Listado 2-22: Complementando el elemento `<article>`

El Listado 2-22 expande el elemento **<article>** utilizado en ejemplos anteriores. Este elemento ahora incluye un elemento **<header>** con el título del artículo y la fecha de publicación, y el texto se ha resaltado con los elementos **<mark>** y **<cite>**. El elemento **<mark>** resalta partes del texto que originalmente no se consideraban importantes, pero que al momento se han vuelto relevantes (quizás debido a que el usuario realizó una búsqueda con ese texto), y el elemento **<cite>**, en este caso, resalta el nombre del autor. Por defecto, los navegadores asignan estilos al texto dentro del elemento **<mark>** que incluyen un fondo amarillo y muestran el contenido del elemento **<cite>** en cursiva, tal como ilustra la siguiente figura.

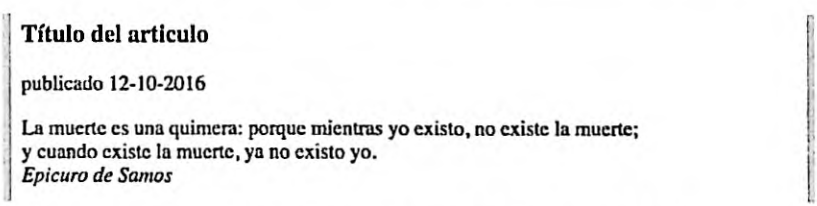

Figura 2-9: Complementando el elemento `<article>`

IMPORTANTE: el valor del atributo `datetime` del elemento `<time>` se debe declarar en formato de ordenador. Este formato requiere la sintaxis `2016-10-12T12:10:45`, donde la T se puede reemplazar por un espacio en blanco y la parte menos significativa se puede ignorar (por ejemplo, `2016-10-12`).

Lo básico: el atributo `pubdate` es un atributo booleano. Este tipo de atributos no requieren un valor; representan el valor `true` (verdadero) cuando están presentes o `false` (falso) en caso contrario.

El resto de los elementos mencionados arriba también se combinan con otros elementos para complementar sus contenidos. Por ejemplo, los elementos `<address>` y `<small>` se insertan normalmente dentro de un elemento `<footer>` para resaltar información acerca de la página o una sección.

```
<footer>
  <address>Toronto, Canada</address>
  <small>&copy; Derechos Reservados 2016</small>
</footer>
```

***Listado 2-23:** Complementando el elemento* `<footer>`

Toronto, Canada
© Derechos Reservados 2016

***Figura 2-10:** Complementando el elemento* `<footer>`

El elemento `<code>` también trabaja junto con otros elementos para presentar contenido, pero tiene una relación particular con el elemento `<pre>`. Estos elementos se implementan juntos para presentar código de programación. El elemento `<code>` indica que el contenido es código de programación y el elemento `<pre>` formatea ese contenido para mostrarlo en pantalla como se ha declarado originalmente en el documento (respetando los espacios y los saltos de línea).

```
<article>
  <pre>
    <code>
function cambiarColor() {
  document.body.style.backgroundColor = "#0000FF";
}
document.addEventListener("click", cambiarColor);
    </code>
  </pre>
</article>
```

***Listado 2-24:** Presentando código con los elementos* `<code>` *y* `<pre>`

El Listado 2-24 define un elemento `<article>` que muestra código programado en JavaScript. Debido a que este código no se encuentra dentro de un elemento `<script>`, el navegador no lo ejecuta y, debido a que se incluye dentro de un elemento `<pre>`, se presenta con los saltos de línea y los espacios declarados en el documento.

```
function cambiarColor() {
  document.body.style.backgroundColor = "#0000FF";
}
document.addEventListener("click", cambiarColor);
```

Figura 2-11: Código de programación presentado con los elementos <code> *y* <pre>

Lo básico: a veces los nombres de los elementos y su contenido no ofrecen suficiente información al desarrollador para entender el propósito del código. En estas situaciones, HTML nos permite escribir comentarios. Los comentarios son textos que se incluyen en el documento, pero que no son procesados por el navegador. Para agregar un comentario, tenemos que escribir el texto entre las etiquetas `<!--` y `-->`, como en `<!-- Este es un comentario -->`.

Enlaces

Conectar documentos con otros documentos mediante enlaces es lo que hace posible la Web. Como mencionamos anteriormente, un enlace es contenido asociado a una URL que indica la ubicación de un recurso. Cuando el usuario hace clic en el contenido (texto o imagen), el navegador descarga el recurso. HTML incluye el siguiente elemento para crear enlaces.

<a>—Este elemento crea un enlace. El texto o la imagen que representa el enlace se incluye entre las etiquetas de apertura y cierre. El elemento incluye el atributo **href** para especificar la URL del enlace.

Los enlaces se pueden crear para conectar el documento actual con otros documentos en el mismo sitio web o en otros sitios. Por ejemplo, podemos enlazar las opciones en el menú de nuestra página web a otros documentos en nuestro servidor.

```
<nav>
  <a href="index.html">Principal</a> |
  <a href="fotos.html">Fotos</a> |
  <a href="videos.html">Videos</a> |
  <a href="contacto.html">Contacto</a>
</nav>
```

Listado 2-25: Enlazando el documento a otros documentos con el elemento <a>

El elemento **<nav>** en el Listado 2-25 incluye cuatro elementos **<a>** por cada opción del menú. Los elementos incluyen el atributo **href** para indicar al navegador el documento que tiene que abrir cuando el usuario hace clic en el enlace. Por defecto, los enlaces se muestran subrayados y en color azul (o violeta si el usuario ya ha hecho clic en ellos).

Principal | Fotos | Videos | Contacto

Figura 2-12: Hipervinculos

Cuando el usuario hace clic en cualquiera de estos enlaces, el navegador descarga el documento indicado por el atributo **href** y muestra su contenido en pantalla. Por ejemplo, si hacemos clic en el enlace creado para la opción **Contacto** en el código del Listado 2-25, el navegador descarga el archivo contacto.html y muestra su contenido en la pantalla, como ilustra la Figura 2-13 (este ejemplo asume que hemos creado un archivo llamado contacto.html).

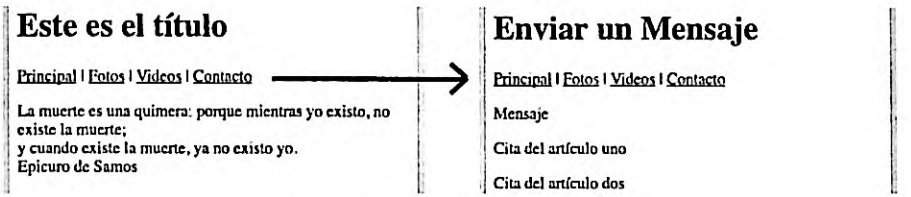

Figura 2-13: Navegando entre documentos

Hágalo usted mismo: cree un nuevo archivo llamado contacto.html. Copie el código HTML del archivo index.html en el archivo contacto.html. Reemplace el elemento **<nav>** en ambos archivos con el código del Listado 2-25. Cambie el título en el elemento **<header>** del archivo contacto.html por el texto «Enviar un Mensaje». Abra el archivo index.html en su navegador y haga clic en la opción **Contacto**. El navegador debería abrir el archivo contacto.html y mostrarlo en pantalla, según ilustra la Figura 2-13.

Los documentos enlazados en el menú del Listado 2-25 pertenecen al mismo sitio web y esa es la razón por la que usamos URL relativas para especificar su ubicación, pero si lo que necesitamos es crear enlaces a documentos que no están almacenados en nuestro servidor, tenemos que usar URL absolutas, como en el siguiente ejemplo.

```
<footer>
  <address>Toronto, Canada</address>
  <small>&copy; 2016 <a href="http://www.jdgauchat.com">J.D
Gauchat</a></small>
</footer>
```

Listado 2-26: Enlazando el documento a documentos en otros sitios web con el elemento <a>

El código en el Listado 2-26 agrega un enlace al pie de página de nuestro ejemplo que apunta al sitio web www.jdgauchat.com. El enlace trabaja como cualquier otro, pero ahora el navegador tiene la URL completa para acceder al documento (en este caso, el archivo index del sitio web con el domino www.jdgauchat.com).

Figura 2-14: Enlace a un documento externo

El elemento <a> puede incluir el atributo **target** para especificar el destino en el cual el documento será abierto. El valor **_self** se asigna por defecto, lo que significa que el documento se abre en la misma ubicación que el documento actual (el mismo recuadro o ventana). Otros valores son **_blank** (el documento se abre en una nueva ventana), **_parent** (el documento se abre en el recuadro padre), y **_top** (el documento se abre en la ventana actual).

Como veremos más adelante, los documentos se pueden abrir en recuadros insertados dentro de otros documentos. El valor del atributo **target** considera esta jerarquía de recuadros, pero debido a que los recuadros no se usan de forma frecuente en sitios web modernos, los dos valores más comunes son **_self**, para abrir el documento en la misma ventana, y **_blank**, para abrir el documento en una nueva ventana. El siguiente ejemplo implementa el último valor para acceder al dominio www.jdgauchat.com desde una nueva ventana, de modo que el usuario nunca abandona nuestro sitio web.

```
<footer>
  <address>Toronto, Canada</address>
  <small>&copy; 2016 <a href="http://www.jdgauchat.com"
target="_blank">J.D Gauchat</a></small>
</footer>
```

***Listado 2-27:** Abriendo un enlace en una nueva ventana*

Además de conectar un documento con otro, los enlaces también se pueden crear hacia otros elementos dentro del mismo documento. Esto es particularmente útil cuando el documento genera una página extensa que el usuario debe desplazar para poder ver todo su contenido. Aprovechando esta característica, podemos crear enlaces hacia diferentes partes de una página. Cuando el usuario quiere ver algo que no es visible al momento, puede hacer clic en estos enlaces y el navegador desplaza la página hasta que el elemento apuntado por el enlace aparece en la pantalla. El elemento que queremos enlazar tiene que ser identificado con el atributo **id**. Para crear un enlace a un elemento, debemos incluir el valor asignado a este atributo precedido por el carácter #, igual que ilustra el siguiente ejemplo.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Este texto es el título del documento</title>
  <meta charset="utf-8">
  <meta name="description" content="Este es un documento HTML5">
  <meta name="keywords" content="HTML, CSS, JavaScript">
  <link rel="stylesheet" href="misestilos.css">
</head>
<body>
  <header id="titulo">
    Este es el título
  </header>
  <nav>
    Principal | Fotos | Videos | Contacto
  </nav>
  <main>
    <section>
      <p>Artículo 1</p>
```

```
      <p>Artículo 2</p>
      <p>Artículo 3</p>
      <p>Artículo 4</p>
      <p><a href="#titulo">Volver</a></p>
    </section>
  </main>
  <footer>
    &copy; Derechos Reservados 2016
  </footer>
</body>
</html>
```

Listado 2-28: Creando enlaces a elementos en el mismo documento

En el documento del Listado 2-28, usamos el atributo `id` con el valor "titulo" para identificar el elemento `<header>`. Usando este valor y el carácter #, creamos un enlace al final del contenido que lleva al usuario hacia la parte superior de la página. Cuando el usuario hace clic en el enlace, en lugar de abrir un documento, el navegador desplaza la página hasta que el contenido del elemento `<header>` se vuelve visible.

Hágalo usted mismo: actualice el código en su archivo index.html con el código del Listado 2-28. Agregue más párrafos con elementos `<p>` para generar más contenido dentro del elemento `<section>`. Abra el documento en su navegador, desplace la página hacia abajo, y haga clic en el enlace **Volver**. Si la cabecera no es visible, el navegador desplazará la página hacia arriba para mostrarla en pantalla.

El elemento `<a>` también se puede usar para crear enlaces a aplicaciones. HTML ofrece las palabras clave `mailto` y `tel` para especificar una cuenta de correo o un número de teléfono. Cuando se hace clic en un enlace de estas características, el sistema abre el programa encargado de responder a este tipo de solicitudes (enviar un correo o hacer una llamada telefónica) y le envía los datos especificados en el enlace. El siguiente ejemplo implementa la palabra clave `mailto` para enviar un correo electrónico.

```
<footer>
  <address>Toronto, Canada</address>
  <small>&copy; 2016 <a href="mailto:info@jdgauchat.com">J.D
Gauchat</a></small>
</footer>
```

Listado 2-29: Enviando un correo electrónico

Hágalo usted mismo: reemplace el elemento `<footer>` en su archivo index.html con el código del Listado 2-29. Abra el documento en su navegador y haga clic en el enlace. El sistema debería abrir su programa de correo para enviar un mensaje a la cuenta info@jdgauchat.com.

Los documentos a los que se accede a través de un enlace creado por el elemento `<a>` son descargados por el navegador y mostrados en pantalla, pero a veces los usuarios no necesitan

que el navegador abra el documento, sino que el archivo se almacene en sus discos duros para usarlo más adelante. HTML ofrece dos atributos para este propósito:

download—Este es un atributo booleano que, cuando se incluye, indica que en lugar de leer el archivo el navegador debería descargarlo.

ping—Este atributo declara la ruta del archivo que se debe abrir en el servidor cuando el usuario hace clic en el enlace. El valor puede ser una o más URL separadas por un espacio.

Cuando el atributo `download` se encuentra presente dentro de un elemento `<a>`, el archivo especificado por el atributo `href` se descarga y almacena en el disco duro del usuario. Por otro lado, el archivo que indica el atributo `ping` no se descarga, sino que se ejecuta en el servidor. Este archivo se puede usar para ejecutar código que almacena información en el servidor cada vez que el archivo principal se descarga o para llevar un control de las veces que esta acción ocurre. En el siguiente ejemplo, implementamos ambos atributos para permitir al usuario descargar un archivo PDF.

```
<article>
  <p>La muerte es una quimera: porque mientras yo existo, no existe la
muerte;<br>y cuando existe la muerte, ya no existo yo.<br>Epicuro de
Samos</p>
  <footer>
    <a href="http://www.formasterminds.com/content/miarchivo.pdf"
ping="http://www.formasterminds.com/control.php" download>Clic aquí
para descargar</a>
  </footer>
</article>
```

***Listado 2-30:** Aplicando los atributos* `ping` *y* `download`

En el ejemplo del Listado 2-30, agregamos un pie de página al artículo de ejemplos anteriores con un enlace a un archivo PDF. En circunstancias normales, un navegador moderno mostraría el contenido del archivo en pantalla, pero en este caso el atributo `download` obliga al navegador a descargar el archivo y almacenarlo en el disco duro.

Este ejemplo incluye un atributo `ping` que apunta a un archivo llamado control.php. Como resultado, cada vez que el usuario hace clic en el enlace, se descarga el archivo PDF y el código PHP se ejecuta en el servidor, lo que permite al desarrollador hacer un seguimiento de las veces que esta acción ocurre (almacenando información acerca del usuario en una base de datos, por ejemplo).

Hágalo usted mismo: reemplace el elemento `<article>` en su archivo index.html por el código del Listado 2-30 y abra el documento en su navegador. Cuando haga clic en el enlace, el navegador debería descargar el archivo PDF. Elimine el atributo `download` para comparar el comportamiento del navegador.

IMPORTANTE: el propósito del atributo `ping` es ejecutar código en el servidor. En el ejemplo hemos usado un archivo con código PHP, pero podría hacer lo mismo con cualquier otro lenguaje de programación que funcione en el servidor, como Python o Ruby. El modo de programar el archivo

asignado al atributo `ping` depende del lenguaje que usemos y lo que queramos lograr. El tema va más allá del propósito de este libro. Para obtener más información sobre PHP, visite nuestro sitio web y siga los enlaces de este capítulo.

Imágenes

Las imágenes pueden se pueden considerar el segundo medio más importante en la Web. HTML incluye los siguientes elementos para introducir imágenes en nuestros documentos.

<img>—Este elemento inserta una imagen en el documento. El elemento requiere del atributo `src` para especificar la URL del archivo con la imagen que queremos incorporar.

<picture>—Este elemento inserta una imagen en el documento. Trabaja junto con el elemento `<source>` para ofrecer múltiples imágenes en diferentes resoluciones. Es útil para crear sitios web adaptables, como veremos en el Capítulo 5.

<figure>—Este elemento representa contenido asociado con el contenido principal, pero que se puede eliminar sin que se vea afectado, como fotos, vídeos, etc.

<figcaption>—Este elemento introduce un título para el elemento `<figure>`.

Para incluir una imagen en el documento, solo necesitamos declarar el elemento `<img>` y asignar la URL del archivo al atributo `src`.

```
<article>
  <p>La muerte es una quimera: porque mientras yo existo, no existe la
muerte;<br>y cuando existe la muerte, ya no existo yo.<br>Epicuro de
Samos</p>
  <img src="miimagen.jpg">
</article>
```

Listado 2-31: Incluyendo una imagen en el documento

El código del Listado 2-31 carga la imagen del archivo miimagen.jpg y la muestra en pantalla en su tamaño original, como lo ilustra la Figura 2-15.

Figura 2-15: Imagen en el documento

La imagen se representa en su tamaño original, pero podemos definir un tamaño personalizado y algunos parámetros de configuración usando el resto de los atributos disponibles para el elemento `<img>`.

width—Este atributo declara el ancho de la imagen.

height—Este atributo declara la altura de la imagen.

alt—Este atributo especifica el texto que se muestra cuando la imagen no se puede cargar.

srcset—Este atributo nos permite especificar una lista de imágenes de diferentes resoluciones que el navegador puede cargar para el mismo elemento.

sizes—Este atributo especifica una lista de *media queries* (consulta de medios) junto con distintos tamaños de imágenes para que el navegador decida qué mostrar según la resolución de la pantalla. Estudiaremos *media queries* y diseño web adaptable en el Capítulo 5.

crossorigin—Este atributo establece las credenciales para imágenes de origen cruzado (múltiples orígenes). Los valores posible son **anonymous** (sin credenciales) y **use-credentials** (requiere credenciales). Estudiaremos la tecnología CORS y cómo trabajar con imágenes procedentes de diferentes orígenes en el Capítulo 11.

Los atributos **width** y **height** determinan las dimensiones de la imagen, pero no tienen en cuenta la relación. Si declaramos ambos valores sin considerar la proporción original de la imagen, el navegador deberá estirar o achatar la imagen para adaptarla a las dimensiones definidas. Para reducir la imagen sin cambiar la proporción original, podemos especificar uno solo de los atributos y dejar que el navegador calcule el otro.

```
<article>
  <p>La muerte es una quimera: porque mientras yo existo, no existe la
muerte;<br>y cuando existe la muerte, ya no existo yo.<br>Epicuro de
Samos</p>
  <img src="miimagen.jpg" width="150" alt="Arboles en mi patio">
</article>
```

Listado 2-32: Reduciendo el tamaño de la imagen

El código en el Listado 2-32 agrega el atributo **width** con el valor 150 al elemento `<img>` introducido en el ejemplo anterior. Esto reduce el ancho de la imagen a 150 píxeles, pero como el atributo **height** no se ha declarado, la altura de la imagen se calcula automáticamente considerando las proporciones originales de la imagen. El resultado se muestra en la Figura 2-16.

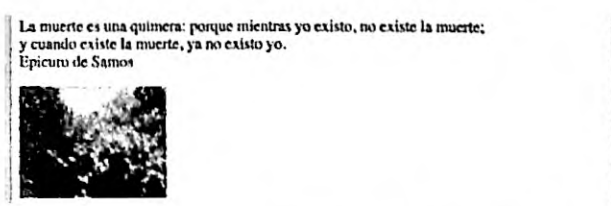

Figura 2-16: Imagen con un tamaño personalizado

Hágalo usted mismo: descargue la imagen miimagen.jpg desde nuestro sitio web. Reemplace el elemento `<article>` en su archivo index.html por el código del Listado 2-31 y abra el documento en su navegador. Debería ver algo similar a la Figura 2-15. Repita el proceso con el código del Listado 2-32. Ahora debería ver la imagen reducida, según se ilustra en la Figura 2-16. El elemento `<img>` en el código del Listado 2-32 también incluye el atributo `alt`. Para probar este atributo, borre el archivo miimagen.jpg y actualice el documento en su navegador. Como el navegador ya no puede encontrar el archivo de la imagen, mostrará el texto asignado al atributo `alt` en su lugar.

Algunas imágenes, como los iconos, son importantes porque otorgan un significado al resto del contenido, pero otras, como la imagen de estos ejemplos, actúan como complemento y se pueden eliminar sin que afecten al flujo de información. Cuando esta clase de información se encuentra presente, se puede utilizar el elemento `<figure>` para identificarla. Este elemento se suele implementar junto con el elemento `<figcaption>` para incluir texto descriptivo. En el siguiente ejemplo usamos estos dos elementos para identificar nuestra imagen y mostrar su título al usuario.

```
<article>
  <p>La muerte es una quimera: porque mientras yo existo, no existe la
muerte;<br>y cuando existe la muerte, ya no existo yo.<br>Epicuro de
Samos</p>
  <figure>
    <img src="miimagen.jpg" width="150" alt="Arboles en mi patio">
    <figcaption>Arboles en mi patio<figcaption>
  </figure>
</article>
```

***Listado 2-33:** Identificando la imagen con el elemento* `<figure>`

Por defecto, los navegadores asignan márgenes laterales al elemento `<figure>`. El resultado se muestra en la Figura 2-17.

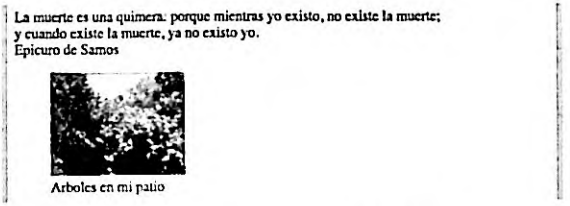

***Figura 2-17:** Imagen junto a su título*

Listados

A menudo, la información se debe representar como una lista de ítems. Por ejemplo, muchos sitios web incluyen listados de libros, películas, o términos y descripciones. Para crear estos listados, HTML ofrece los siguientes elementos.

**
—Este elemento crea una lista de ítems sin orden. Está compuesto por etiquetas de apertura y cierre para agrupar los ítems (
** y **
) y trabaja junto con el elemento **<li> para definir cada uno de los ítems de la lista.

**
—Este elemento crea una lista ordenada de ítems. Está compuesto por etiquetas de apertura y cierre para agrupar los ítems (
** y **
) y trabaja junto con el elemento **<li> para definir los ítems de la lista. Este elemento puede incluir los atributos **reversed** para invertir el orden de los indicadores, **start** para determinar el valor desde el cual los indicadores tienen que comenzar a contar y **type** para determinar el tipo de indicador que queremos usar. Los valores disponibles para el atributo **type** son **1** (números), **a** (letras minúsculas), **A** (letras mayúsculas), **i** (números romanos en minúsculas) e **I** (números romanos en mayúsculas).

<dl>—Este elemento crea una lista de términos y descripciones. El elemento trabaja junto con los elementos **<dt>** y **<dd>** para definir los ítems de la lista. El elemento **<dl>** define la lista, el elemento **<dt>** define los términos y el elemento **<dd>** define las descripciones.

El elemento que usamos para crear la lista depende de las características del contenido. Por ejemplo, si el orden de los ítems no es importante, podemos usar el elemento **<ul>**. En esta clase de listas, los ítems se declaran entre las etiquetas **<ul>** con el elemento **<li>**, como muestra el siguiente ejemplo.

```
<aside>
  <ul>
    <li>IT, Stephen King</li>
    <li>Carrie, Stephen King</li>
    <li>El Resplandor, Stephen King</li>
    <li>Misery, Stephen King</li>
  </ul>
</aside>
```

Listado 2-34: Creando una lista de ítems sin orden

El elemento **<ul>** presenta los ítems en el orden en el que se han declarado en el código y los identifica con un punto del lado izquierdo.

- IT, Stephen King
- Carrie, Stephen King
- El Resplandor, Stephen King
- Misery, Stephen King

Figura 2-18: Lista sin orden

Si necesitamos declarar la posición de cada ítem, podemos crear la lista con el elemento **<ol>**. Este elemento crea una lista de ítems en el orden en el que se han declarado en el código, pero en lugar de usar puntos para identificarlos, les asigna un valor. Por defecto, los indicadores se crean con números, pero podemos cambiarlos con el atributo **type**. En el siguiente ejemplo, usamos letras mayúsculas.

```
<aside>
  <ol type="A">
    <li>IT, Stephen King</li>
    <li>Carrie, Stephen King</li>
    <li>El Resplandor, Stephen King</li>
    <li>Misery, Stephen King</li>
  </ol>
</aside>
```

Listado 2-35: Creando una lista ordenada de ítems

Figura 2-19: Lista ordenada

Aunque los ítems se muestran siempre en el orden en el que se han declarado en el código, podemos utilizar otros atributos del elemento `<ol>` para cambiar el orden de los indicadores. Por ejemplo, agregando el atributo `reversed` logramos que los indicadores se muestren en orden invertido.

```
<aside>
  <ol type="A" reversed>
    <li>IT, Stephen King</li>
    <li>Carrie, Stephen King</li>
    <li>El Resplandor, Stephen King</li>
    <li>Misery, Stephen King</li>
  </ol>
</aside>
```

Listado 2-36: Creando una lista en orden invertido

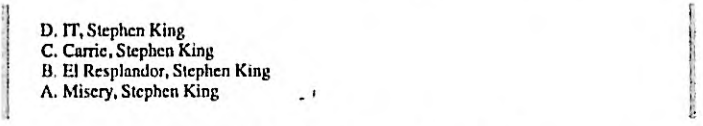

Figura 2-20: Lista en orden invertido

Los elementos `<dl>`, `<dt>`, y `<dd>` trabajan de forma similar al elemento `<ul>`, pero su propósito es mostrar una lista de términos y descripciones. Los términos se representan por el elemento `<dt>` y las descripciones por el elemento `<dd>`. En el siguiente ejemplo, los usamos para agregar la descripción de los libros listados anteriormente.

```
<aside>
  <dl>
    <dt>IT</dt>
```

```
    <dd>Tras lustros de tranquilidad y lejania una antigua promesa
infantil les hace volver al lugar en el que vivieron su infancia y
juventud como una terrible pesadilla.</dd>
    <dt>Carrie</dt>
    <dd>Sus compañeros se burlan de ella, pero Carrie tiene un don.
Puede mover cosas con su mente. Este es su poder y su gran
problema.</dd>
    <dt>El Resplandor</dt>
    <dd>Al escritor Jack Torrance le es ofrecido un empleo como cuidador
del hotel Overlook durante el invierno junto a su familia.</dd>
    <dt>Misery</dt>
    <dd>Paul Sheldon es un famoso escritor de novelas románticas
ambientadas en la época victoriana, cuyo personaje principal se llama
Misery Chastain.</dd>
  </dl>
</aside>
```

Listado 2-37: Creando una lista de términos y descripciones

Por defecto, los navegadores otorgan estilos a este tipo de listas que incluyen márgenes a los lados para diferenciar los términos de las descripciones. El resultado se muestra en la Figura 2-21.

IT
Tras lustros de tranquilidad y lejania una antigua promesa infantil les hace volver al lugar en el que vivieron su infancia y juventud como una terrible pesadilla.
Carrie
Sus compañeros se burlan de ella, pero Carrie tiene un don. Puede mover cosas con su mente. Este es su poder y su gran problema.
El Resplandor
Al escritor Jack Torrance le es ofrecido un empleo como cuidador del hotel Overlook durante el invierno junto a su familia.
Misery
Paul Sheldon es un famoso escritor de novelas románticas ambientadas en la época victoriana, cuyo personaje principal se llama Misery Chastain.

Figura 2-21: Lista de términos y descripciones

Hágalo usted mismo: reemplace el elemento `<aside>` en su archivo index.html con el ejemplo que quiere probar y abra el documento en su navegador. Inserte el atributo `start` con el valor "3" en el elemento `<ol>` del Listado 2-35. Los indicadores deberían comenzar a contar desde la letra C.

Los siguientes elementos se han diseñado con propósitos diferentes, pero también se utilizan frecuentemente para construir listas de ítems.

<blockquote>—Este elemento representa un bloque de texto que incluye una cita tomada de otro texto en el documento.

<details>—Este elemento crea una herramienta que se expande cuando se hace clic en ella para mostrar información adicional. La parte visible se define con el elemento `<summary>`, y se pueden usar elementos comunes como `<p>` para definir el contenido.

El elemento `<blockquote>` es similar al elemento `<p>`, pero como también incluye márgenes a los costados, se ha usado tradicionalmente para presentar listas de valores, como lo demuestra el siguiente ejemplo.

```
<aside>
  <blockquote>IT</blockquote>
  <blockquote>Carrie</blockquote>
  <blockquote>El Resplandor</blockquote>
  <blockquote>Misery</blockquote>
</aside>
```

Listado 2-38: Creando una lista con el elemento `<blockquote>`

El listado generado por el elemento **<blockquote>** se muestra igual que otras listas, pero no incluye puntos o números para identificar cada ítem.

Figura 2-22: Listado de ítems con el elemento `<blockquote>`

En sitios web modernos son comunes las herramientas que revelan información adicional cuando el usuario lo requiere. Para ofrecer esta posibilidad, HTML incluye el elemento **<details>**. Este elemento muestra un título, especificado por el elemento **<summary>**, y contenido que se puede representar por elementos comunes como **<p>** o **<blockquote>**. Debido a esto, el elemento **<details>** se puede utilizar para revelar una lista de valores, como lo hacemos en el siguiente ejemplo.

```
<details>
  <summary>My Books</summary>
  <p>IT</p>
  <p>Carrie</p>
  <p>El Resplandor</p>
  <p>Misery</p>
</details>
```

Listado 2-39: Revelando información con los elementos `<details>` *y* `<summary>`

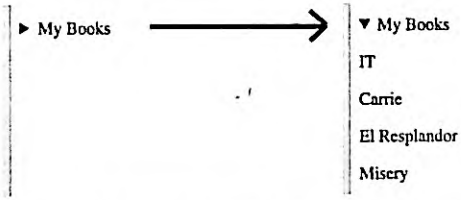

Figura 2-23: El elemento `<details>` *antes y después de ser cliqueado*

Tablas

Las tablas organizan información en filas y columnas. Debido a sus características, se usaron durante mucho tiempo para estructurar documentos HTML, pero con la introducción de CSS, los desarrolladores pudieron lograr el mismo efecto implementando otros elementos. Aunque ya no se recomienda usar tablas para definir la estructura de un documento, todavía se utilizan para presentar información tabular, como estadísticas o especificaciones técnicas, por ejemplo. HTML incluye varios elementos para crear una tabla. Los siguientes son los más utilizados.

<table>—Este elemento define una tabla. Incluye etiquetas de apertura y cierre para agrupar el resto de los elementos que definen la tabla.

<tr>—Este elemento define una fila de celdas. Incluye etiquetas de apertura y cierre para agrupar las celdas.

<td>—Este elemento define una celda. Incluye etiquetas de apertura y cierre para delimitar el contenido de la celda y puede incluir los atributos `colspan` y `rowspan` para indicar cuántas columnas y filas ocupa la celda.

<th>—Este elemento define una celda para la cabecera de la tabla. Incluye etiquetas de apertura y cierre para delimitar el contenido de la celda y puede incluir los atributos `colspan` y `rowspan` para indicar cuántas columnas y filas ocupa la celda.

Para incluir una tabla en el documento, primero tenemos que declarar el elemento `<table>` y luego describir las filas una por una con los elementos `<tr>` y `<td>`, como muestra el siguiente ejemplo.

```
<article>
  <table>
    <tr>
      <td>IT</td>
      <td>Stephen King</td>
      <td>1986</td>
    </tr>
    <tr>
      <td>Carrie</td>
      <td>Stephen King</td>
      <td>1974</td>
    </tr>
    <tr>
      <td>El Resplandor</td>
      <td>Stephen King</td>
      <td>1977</td>
    </tr>
  </table>
</article>
```

Listado 2-40: Creando una tabla

Debido a que el navegador interpreta el documento de forma secuencial desde la parte superior a la inferior, cada vez que declaramos una fila, tenemos que declarar las celdas que

corresponden a esa fila y su contenido. Siguiendo este patrón, en el Listado 2-40, creamos una tabla para mostrar libros, uno por fila. La primer celda de cada fila representa el título del libro, la segunda celda representa el autor, y la tercera celda el año de publicación. Cuando el navegador abre este documento, muestra la información en el orden en el que se ha declarado en el código y con el tamaño determinado por el contenido de las celdas.

IT	Stephen King	1986
Carrie	Stephen King	1974
El Resplandor	Stephen King	1977

Figura 2-24: Tabla con estilos por defecto

Si queremos incluir una cabecera para describir el contenido de cada columna, podemos crear una fila de celdas adicional representadas con el elemento **<th>**.

```
<article>
  <table>
    <tr>
      <th>Título</th>
      <th>Autor</th>
      <th>Año</th>
    </tr>
    <tr>
      <td>IT</td>
      <td>Stephen King</td>
      <td>1986</td>
    </tr>
    <tr>
      <td>Carrie</td>
      <td>Stephen King</td>
      <td>1974</td>
    </tr>
    <tr>
      <td>El Resplandor</td>
      <td>Stephen King</td>
      <td>1977</td>
    </tr>
  </table>
</article>
```

Listado 2-41: Creando una tabla con cabecera

Por defecto, los navegadores muestran las cabeceras con el texto en negrita y centrado.

Título	Autor	Año
IT	Stephen King	1986
Carrie	Stephen King	1974
El Resplandor	Stephen King	1977

Figura 2-25: Cabecera de tabla con estilos por defecto

Las celdas se pueden estirar para que ocupen más de una columna con los atributos **colspan** y **rowspan**. Por ejemplo, podemos usar solo una celda de cabecera para identificar el título y el autor del libro.

```
<article>
  <table>
    <tr>
      <th colspan="2">Libro</th>
      <th>Año</th>
    </tr>
    <tr>
      <td>IT</td>
      <td>Stephen King</td>
      <td>1986</td>
    </tr>
    <tr>
      <td>Carrie</td>
      <td>Stephen King</td>
      <td>1974</td>
    </tr>
    <tr>
      <td>El Resplandor</td>
      <td>Stephen King</td>
      <td>1977</td>
    </tr>
  </table>
</article>
```

***Listado 2-42:** Estirando celdas*

El ejemplo del Listado 2-42 incluye una celda de cabecera con el título **Libro** para las primeras dos columnas. Debido al valor asignado al atributo **colspan**, esta celda se estira para que ocupe el espacio de dos. El resultado se muestra en la Figura 2-26.

Libro		Año
IT	Stephen King	1986
Carrie	Stephen King	1974
El Resplandor	Stephen King	1977

***Figura 2-26:** Celdas de múltiples columnas*

Lo básico: HTML también incluye los elementos **<thead>**, **<tbody>**, y **<tfoot>** para representar la cabecera, el cuerpo, y el pie de la tabla, respectivamente, y otros elementos como **<colgroup>** para agrupar columnas. Para obtener más información, visite nuestro sitio web y siga los enlaces de este capítulo.

Atributos Globales

La mayoría de los navegadores actuales automáticamente traducen el contenido del documento cuando detectan que se ha escrito en un idioma diferente al del usuario, pero en

algunos casos la página puede incluir frases o párrafos enteros que no se deben alterar, como nombres de personas o títulos de películas. Para controlar el proceso de traducción, HTML ofrece un atributo global llamado **translate**. Este atributo puede tomar dos valores: **yes** y **no**. Por defecto, el valor es **yes** (si). En el siguiente ejemplo, usamos un elemento **** para especificar la parte del texto que no se debería traducir.

```
<p>My favorite movie is <span translate="no">Two Roads</span></p>
```

***Listado 2-43:** Usando el atributo* `translate`

El elemento **** se diseñó para presentar texto, pero a diferencia del elemento **<p>** no asigna ningún estilo al contenido, por lo que el texto se presenta en la misma línea y con el tipo de letra y tamaño por defecto. La Figura 2-27 muestra lo que vemos en la ventana del navegador después de traducirlo.

Mi película favorita es Two Roads

***Figura 2-27:** texto traducido por el navegador*

Hágalo usted mismo: inserte el código del Listado 2-43 en la sección principal de su documento y abra el documento en su navegador. Los navegadores como Google Chrome ofrecen una opción para traducir el documento en un menú contextual cuando hacemos clic con el botón derecho del ratón. Si la traducción no se realiza de forma automática, seleccione esta opción para traducir el texto. Una vez traducido el texto, debería ver algo similar a lo que se muestra en la Figura 2-27.

Otro atributo útil que podemos agregar a un elemento HTML es **contenteditable**. Este es un atributo booleano que, si está presente, permite al usuario editar el contenido del elemento. Si el usuario hace clic en un elemento que contiene este atributo, puede cambiar su contenido. El siguiente ejemplo implementa el elemento **** nuevamente para permitir a los usuarios editar el nombre de una película.

```
<p>Mi película favorita es <span contenteditable>Casablanca</span></p>
```

***Listado 2-44:** Usando el atributo* `contenteditable` *para editar contenido*

Hágalo usted mismo: reemplace el párrafo del Listado 2-43 por el párrafo del Listado 2-44 y abra el documento en su navegador. Haga clic en el nombre de la película para cambiarlo.

IMPORTANTE: las modificaciones introducidas por el usuario solo se encuentran disponibles en su ordenador. Si queremos que los cambios se almacenen en el servidor, tenemos que subir esta información con un programa en JavaScript usando Ajax, según veremos en el Capítulo 21.

2.3 Formularios

Los formularios son herramientas que podemos incluir en un documento para permitir a los usuarios insertar información, tomar decisiones, comunicar datos y cambiar el comportamiento de una aplicación. El propósito principal de los formularios es permitir al usuario seleccionar o insertar información y enviarla al servidor para ser procesada. La Figura 2-28 muestra algunas de las herramientas facilitadas este fin.

Figura 2-28: Formulario en el navegador

Definición

Como se muestra en la Figura 2-28, los formularios pueden presentar varias herramientas que permiten al usuario interactuar con el documento, incluidos campos de texto, casillas de control, menús desplegables y botones. Cada una de estas herramientas se representa por un elemento y el formulario queda definido por el elemento `<form>`, que incluye etiquetas de apertura y cierre para agrupar al resto de los elementos y requiere de algunos atributos para determinar cómo se envía la información al servidor.

name—Este atributo especifica el nombre del formulario. También se encuentra disponible para otros elementos, pero es particularmente útil para elementos de formulario, como veremos más adelante.

method—Este atributo determina el método a utilizar para enviar la información al servidor. Existen dos valores disponibles: `GET` y `POST`. El método `GET` se usa para enviar una cantidad limitada de información de forma pública (los datos son incluidos en la URL, la cual no puede contener más de 255 caracteres). Por otro lado, el método `POST` se utiliza para enviar una cantidad ilimitada de información de forma privada (los datos no son visibles al usuario y pueden tener la longitud que necesitemos).

action—Este atributo declara la URL del archivo en el servidor que va a procesar la información enviada por el formulario.

target—Este atributo determina dónde se mostrará la respuesta recibida desde el servidor. Los valores disponibles son `_blank` (nueva ventana), `_self` (mismo recuadro), `_parent` (recuadro padre), y `_top` (la ventana que contiene el recuadro). El valor `_self` se declara por defecto, lo que significa que la respuesta recibida desde el servidor se mostrará en la misma ventana.

enctype—Este atributo declara la codificación aplicada a los datos que envía el formulario. Puede tomar tres valores: `application/x-www-form-urlencoded` (los caracteres son codificados), `multipart/form-data` (los caracteres no son codificados), `text/plain` (solo los espacios son codificados). El primer valor se asigna por defecto.

accept-charset—Este atributo declara el tipo de codificación aplicada al texto del formulario. Los valores más comunes son `UTF-8` e `ISO-8859-1`. El valor por defecto se asigna al documento con el elemento `<meta>` (ver Listado 2-6).

El siguiente ejemplo define un formulario básico. El atributo `name` identifica el formulario con el nombre "formulario", el atributo `method` determina que los datos se incluirán en la URL (GET), y el atributo `action` declara que procesar.php es el archivo que se ejecutará en el servidor para procesar la información y devolver el resultado.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>Formularios</title>
</head>
<body>
  <section>
    <form name="formulario" method="get" action="procesar.php">

    </form>
  </section>
</body>
</html>
```

Listado 2-45: Definiendo un formulario con el elemento `<form>`

Hágalo usted mismo: cree un nuevo archivo HTML en su editor o modifique el archivo del ejemplo anterior con el código del Listado 2-45 y abra el documento en su navegador. En este momento no verá nada en la pantalla porque el formulario se ha declarado vacío. A continuación, desarrollaremos su contenido.

Elementos

Un formulario puede incluir diferentes herramientas para permitir al usuario seleccionar o insertar información. HTML incluye múltiples elementos para crear estas herramientas. Los siguientes son los más utilizados.

`<input>`—Este elemento crea un campo de entrada. Puede recibir diferentes tipos de entradas, dependiendo del valor del atributo `type`.

`<textarea>`—Este elemento crea un campo de entrada para insertar múltiples líneas de texto. El tamaño se puede declarar en números enteros usando los atributos `rows` y `cols`, o en píxeles con estilos CSS, como veremos en el Capítulo 3.

`<select>`—Este elemento crea una lista de opciones que el usuario puede elegir. Trabaja junto con el elemento `<option>` para definir cada opción y el elemento `<optgroup>` para organizar las opciones en grupos.

<button>—Este elemento crea un botón. Incluye el atributo **type** para definir el propósito del botón. Los valores disponibles son **submit** para enviar el formulario (por defecto), **reset** para reiniciar el formulario, y **button** para realizar tareas personalizadas.

<output>—Este elemento representa un resultado producido por el formulario. Se implementa por medio de código JavaScript para mostrar el resultado de una operación al usuario.

<meter>—Este elemento representa una medida o el valor actual de un rango.

<progress>—Este elemento representa el progreso de una operación.

<datalist>—Este elemento crea un listado de valores disponibles para otros controles. Trabaja junto con el elemento **<option>** para definir cada valor.

<label>—Este elemento crea una etiqueta para identificar un elemento de formulario.

<fieldset>—Este elemento agrupa otros elementos de formulario. Se usa para crear secciones dentro de formularios extensos. El elemento puede contener un elemento **<legend>** para definir el título de la sección.

El elemento **<input>** es el más versátil de todos. Este elemento genera un campo de entrada en el que el usuario puede seleccionar o insertar información, pero puede adoptar diferentes características y aceptar varios tipos de valores dependiendo del valor de su atributo **type**. Los siguientes son los valores disponibles para este atributo.

text—Este valor genera un campo de entrada para insertar texto genérico.

email—Este valor genera un campo de entrada para insertar cuentas de correo.

search—Este valor genera un campo de entrada para insertar términos de búsqueda.

url—Este valor genera un campo de entrada para insertar URL.

tel—Este valor genera un campo de entrada para insertar números de teléfono.

number—Este valor genera un campo de entrada para insertar números.

range—Este valor genera un campo de entrada para insertar un rango de números.

date—Este valor genera un campo de entrada para insertar una fecha.

datetime-local—Este valor genera un campo de entrada para insertar fecha y hora.

week—Este valor genera un campo de entrada para insertar el número de la semana (dentro del año).

month—Este valor genera un campo de entrada para insertar el número del mes.

time—Este valor genera un campo de entrada para insertar una hora (horas y minutos).

hidden—Este valor oculta el campo de entrada. Se usa para enviar información complementaria al servidor.

password—Este valor genera un campo de entrada para insertar una clave. Reemplaza los caracteres insertados con estrellas o puntos para ocultar información sensible.

color—Este valor genera un campo de entrada para insertar un color.

checkbox—Este valor genera una casilla de control que permite al usuario activar o desactivar una opción.

radio—Este valor genera un botón de opción para seleccionar una opción de varias posibles.

file—Este valor genera un campo de entrada para seleccionar un archivo en el ordenador del usuario.

button—Este valor genera un botón. El botón trabaja como el elemento `<button>` de tipo `button`. No realiza ninguna acción por defecto; la acción debe ser definida desde JavaScript, como veremos en próximos capítulos.

submit—Este valor genera un botón para enviar el formulario.

reset—Este valor genera un botón para reiniciar el formulario.

image—Este valor carga una imagen que se usa como botón para enviar el formulario. Un elemento `<input>` de este tipo debe incluir el atributo `src` para especificar la URL de la imagen.

Para incluir un formulario en nuestro documento, tenemos que declararlo con el elemento `<form>`, como hemos hecho en el ejemplo anterior, y luego incorporar en su interior todos los elementos que el usuario necesitará para insertar la información y enviarla al servidor. Por ejemplo, si queremos que el usuario inserte su nombre y edad, tenemos que incluir dos campos de entrada para texto y un tercer elemento para crear el botón con el que enviar el formulario.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>Formularios</title>
</head>
<body>
  <section>
    <form name="formulario" method="get" action="procesar.php">
      <p><input type="text" name="nombre"></p>
      <p><input type="text" name="edad"></p>
      <p><input type="submit"></p>
    </form>
  </section>
</body>
</html>
```

Listado 2-46: Incluyendo herramientas en un formulario

La información insertada en el formulario se envía al servidor para ser procesada. Para que el servidor pueda identificar cada valor, los elementos deben incluir el atributo `name`. Con este atributo podemos asignar un nombre único a cada elemento. En el ejemplo del Listado 2-46, llamamos a los campos de entrada "nombre" y "edad" (el elemento `<input>` que crea el botón para enviar el formulario no necesita un nombre porque no envía ningún dato al servidor).

Los elementos de formulario no generan un salto de línea; se muestran en la pantalla uno detrás del otro. Si queremos que el navegador muestre un elemento en cada línea, tenemos que modificar el diseño nosotros mismos. En el Listado 2-46, usamos elementos **<p>** para separar los elementos del formulario, pero este diseño normalmente se logra a través de estilos CSS, como veremos en el Capítulo 4. El resultado se muestra en la Figura 2-29.

***Figura 2-29:** Formulario con dos campos de entrada y un botón para enviar los datos*

Otro atributo que podemos usar en este ejemplo es **value**. El tipo de entrada **submit** crea un botón para enviar el formulario. Por defecto, los navegadores le dan al botón el título **Enviar** (Submit), pero podemos usar el atributo **value** para modificarlo.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>Formularios</title>
</head>
<body>
  <section>
    <form name="formulario" method="get" action="procesar.php">
      <p><input type="text" name="nombre"></p>
      <p><input type="text" name="edad"></p>
      <p><input type="submit" value="Enviar Datos"></p>
    </form>
  </section>
</body>
</html>
```

***Listado 2-47:** Asignando un título diferente para el botón Enviar*

El atributo **value** también se puede usar para declarar el valor inicial de un elemento. Por ejemplo, podemos insertar la edad del usuario en el campo **edad** si ya la conocemos.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>Formularios</title>
</head>
<body>
  <section>
    <form name="formulario" method="get" action="procesar.php">
      <p><input type="text" name="nombre"></p>
      <p><input type="text" name="edad" value="35"></p>
```

```
      <p><input type="submit" value="Enviar Datos"></p>
    </form>
  </section>
</body>
</html>
```

Listado 2-48: Declarando valores iniciales

Hágalo usted mismo: copie el código del Listado 2-48 dentro de su archivo HTML y abra el documento en su navegador. Debería ver un formulario similar al de la Figura 2-29 pero con el número 35 dentro del campo **edad** y el botón para enviar el formulario con el título **Enviar datos**.

Los formularios necesitan incluir descripciones que le indiquen al usuario lo datos que debe introducir. Por esta razón, HTML incluye el elemento **<label>**. Debido a que estos elementos no solo le indican al usuario qué valor debe introducir, sino que además ayudan al navegador a identificar cada parte del formulario, tienen asociarse al elemento al que están describiendo. Para asociar un elemento **<label>** con el elemento de formulario correspondiente, podemos incluir el elemento de formulario dentro del elemento **<label>**, como muestra el siguiente ejemplo.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>Formularios</title>
</head>
<body>
  <section>
    <form name="formulario" method="get" action="procesar.php">
      <p>
        <label>Nombre: <input type="text" name="nombre"></label>
      </p>
      <p>
        <label>Edad: <input type="text" name="edad"></label>
      </p>
      <p><input type="submit" value="Enviar"></p>
    </form>
  </section>
</body>
</html>
```

Listado 2-49: Identificando elementos de formulario

Otra alternativa para asociar un elemento **<label>** con su elemento de formulario es implementando el atributo **for**. El atributo **for** conecta el elemento **<label>** con el elemento de formulario por medio del valor del atributo **id**, según ilustra el siguiente ejemplo.

```
<!DOCTYPE html>
<html lang="es">
<head>
```

```
  <meta charset="utf-8">
  <title>Formularios</title>
</head>
<body>
  <section>
    <form name="formulario" method="get" action="procesar.php">
      <p>
        <label for="nombre">Nombre: </label>
        <input type="text" name="nombre" id="nombre"></label>
      </p>
      <p>
        <label for="edad">Edad: </label>
        <input type="text" name="edad" id="edad"></label>
      </p>
      <p><input type="submit" value="Enviar"></p>
    </form>
  </section>
</body>
</html>
```

Listado 2-50: *Asociando etiquetas con elementos por medio del atributo* `for`

El elemento **<label>** no incluye ningún estilo por defecto; lo único que hace es asociar una etiqueta con un elemento y, por lo tanto, el texto se muestra en pantalla con el tipo de letra y el tamaño por defecto.

Figura 2-30: *Elementos identificados con una etiqueta*

Los campos de entrada usados en los ejemplos anteriores eran de tipo **text**, lo que significa que los usuarios pueden introducir cualquier clase de texto que deseen, pero esto no es lo que necesitamos para este formulario. El primer campo espera un nombre, por lo que no debería permitir que se introduzcan números o textos muy extensos, y el segundo campo espera la edad del usuario, por lo que no debería aceptar ningún tipo de carácter excepto números. Para determinar cuántos caracteres se pueden introducir, el elemento **<input>** debe incluir los siguientes atributos.

maxlength—Este atributo especifica el máximo número de caracteres que se permite introducir en el campo.

minlength—Este atributo especifica el mínimo número de caracteres que se permite introducir en el campo.

El siguiente ejemplo limita el nombre a un máximo de 15 caracteres.

```
<!DOCTYPE html>
<html lang="es">
```

```
<head>
  <meta charset="utf-8">
  <title>Formularios</title>
</head>
<body>
  <section>
    <form name="formulario" method="get" action="procesar.php">
      <p><label>Nombre: <input type="text" name="nombre"
maxlength="15"></label></p>
      <p><label>Edad: <input type="text" name="edad"></label></p>
      <p><input type="submit" value="Enviar"></p>
    </form>
  </section>
</body>
</html>
```

Listado 2-51: Declarando el máximo número de caracteres permitidos

El atributo **maxlength** implementado en el formulario del Listado 2-51 limita el número de caracteres que el usuario puede introducir, pero el tipo de campo es aún **text**, lo que significa que en el campo se puede escribir cualquier valor. Si el usuario escribe un número en el campo **nombre** o letras en el campo **edad**, el navegador considerará la entrada válida. Para controlar lo que el usuario puede introducir, tenemos que declarar un tipo de campo diferente con el atributo **type**. El siguiente ejemplo declara el tipo **number** para el campo **edad** para permitir que solo se introduzcan números, e incluye otros campos para que el usuario pueda declarar su cuenta de correo, número de teléfono y sitio web.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>Formularios</title>
</head>
<body>
  <section>
    <form name="formulario" method="get" action="procesar.php">
      <p><label>Nombre: <input type="text" name="nombre"
maxlength="15"></label></p>
      <p><label>Edad: <input type="number" name="edad"></label></p>
      <p><label>Correo: <input type="email" name="correo"></label></p>
      <p><label>Teléfono: <input type="tel" name="telefono"></label></p>
      <p><label>Sitio Web: <input type="url" name="sitioweb"></label></p>
      <p><input type="submit" value="Enviar"></p>
    </form>
  </section>
</body>
</html>
```

Listado 2-52: Solicitando tipos de entrada específicos

El tipo **number** asignado al campo **edad** en el Listado 2-52 le dice al elemento que solo acepte números. El resto de los tipos de entrada implementados en este ejemplo no imponen ninguna restricción en los caracteres introducidos, pero le indican al navegador la clase de valores que se esperan del usuario. Por ejemplo, el tipo **email** espera una cuenta de correo,

de modo que si el dato introducido no es una cuenta de correo, el navegador no permite que se envíe el formulario y muestra un error en pantalla. El tipo **url** trabaja exactamente igual que el tipo **email**, pero con direcciones web. Este tipo de campo acepta solo URL absolutas y devuelve un error si el valor no es válido. Otros tipos como **tel** no exigen ninguna sintaxis en particular pero le solicitan al navegador que sugiera al usuario posibles valores a introducir o incluya un teclado específico en los dispositivos que lo requieren (en dispositivos móviles, el teclado que se muestra cuando el usuario hace clic en el campo **telefono** incluye solo dígitos para facilitar que se introduzcan números telefónicos.

Aunque estos tipos de campo presentan sus propias restricciones, todos se ven iguales en el navegador.

***Figura 2-31:** Campos de entrada de diferentes tipos*

Como ilustra la Figura 2-31, por defecto los navegadores incluyen flechas en el lado derecho de un campo de tipo **number** con las que podemos seleccionar un número. Para establecer restricciones en los números que el usuario puede seleccionar con estas flechas o controlar los números permitidos, los elementos **<input>** de este tipo pueden incluir los siguientes atributos.

min—El valor de este atributo determina el valor mínimo que acepta el campo.

max—El valor de este atributo determina el valor máximo que acepta el campo.

step—El valor de este atributo determina el número por el cual el valor del campo se puede incrementar o reducir. Por ejemplo, si declaramos el valor 5 para este atributo y un valor mínimo de 0 y uno máximo de 10 para el campo, el navegador no nos dejará introducir valores entre 0 y 5 o 5 y 10.

El siguiente ejemplo restringe el valor insertado en el campo **edad** de nuestro formulario a un mínimo de 13 y un máximo de 100.

```
<input type="number" name="edad" min="13" max="100">
```

***Listado 2-53:** Restringiendo los números*

Otro tipo de entrada que implementa estos mismo atributos es **range**. El tipo **range** crea un campo que nos permite seleccionar un número desde un rango de valores. El valor inicial se establece de acuerdo a los valores de los atributos **min** y **max**, pero podemos declarar un valor específico con el atributo **value**, como muestra el siguiente ejemplo.

```
<input type="range" name="edad" min="13" max="100" value="35">
```

Listado 2-54: Implementando el tipo `range`

Los navegadores muestran un campo de entrada de tipo **range** como un control que el usuario puede deslizar para seleccionar un valor.

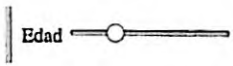

Figura 2-32: Campo de entrada de tipo `range`

Hágalo usted mismo: reemplace el elemento **<input>** en su archivo HTML con el elemento de los Listados 2-53 o 2-54 para probar estos ejemplos. Abra el documento en su navegador e inserte o seleccione un número para ver cómo trabajan estos tipos de entrada. Intente enviar el formulario con un valor menor o mayor a los permitidos. El navegador debería mostrarle un error.

Los valores para el tipo **range** implementado en el ejemplo anterior no se introducen en un campo de texto, sino que se seleccionan desde una herramienta visual generada por el navegador. El tipo **range** no es el único que presenta esta clase de herramientas. Por ejemplo, el tipo **radio** crea un botón circular que se resalta cuando se selecciona (ver Figura 2-28). Esto nos permite crear una lista de valores que el usuario puede seleccionar con solo hacer clic en el botón correspondiente. Para ofrecer al usuario todas las opciones disponibles, tenemos que insertar un elemento **<input>** por cada opción. Los elementos **<input>** se asocian entre ellos por medio del valor del atributo **name**, y el valor de cada opción se define por el atributo **value**, como muestra el siguiente ejemplo.

```
<form name="formulario" method="get" action="procesar.php">
  <p><label>Nombre: <input type="text" name="nombre"
maxlength="15"></label></p>
  <p><label><input type="radio" name="edad" value="15" checked> 15
Años</label></p>
  <p><label><input type="radio" name="edad" value="30"> 30
Años</label></p>
  <p><label><input type="radio" name="edad" value="45"> 45
Años</label></p>
  <p><label><input type="radio" name="edad" value="60"> 60
Años</label></p>
  <p><input type="submit" value="Enviar"></p>
</form>
```

Listado 2-55: Implementando el tipo `radio`

En el formulario del Listado 2-55 declaramos cuatro elementos **<input>** de tipo **radio** para ofrecer distintas edades que el usuario puede elegir. Como todos los elementos tienen el mismo nombre (**edad**), se consideran parte del mismo grupo y, por lo tanto, solo una de las opciones

puede ser seleccionada a la vez. Además del atributo **name**, también implementamos un atributo booleano llamado **checked**. Este atributo le dice al navegador que seleccione el botón cuando se carga el documento, lo cual determina la edad de 15 años como el valor por defecto.

Figura 2-33: Campos de entrada de tipo `radio`

Como ya mencionamos, solo se puede seleccionar uno de estos botones a la vez. El valor asignado al atributo **value** del elemento seleccionado es el que se enviará al servidor. El tipo **checkbox** genera un tipo de entrada similar. En este caso, el usuario puede seleccionar múltiples valores haciendo clic en las casillas correspondientes.

```
<form name="formulario" method="get" action="procesar.php">
  <p><label>Nombre: <input type="text" name="nombre"
maxlength="15"></label></p>
  <p><label><input type="checkbox" name="edad15" value="15" checked> 15
Años</label></p>
  <p><label><input type="checkbox" name="edad30" value="30" checked> 30
Años</label></p>
  <p><label><input type="checkbox" name="edad45" value="45"> 45
Años</label></p>
  <p><label><input type="checkbox" name="edad60" value="60"> 60
Años</label></p>
  <p><input type="submit" value="Enviar"></p>
</form>
```

Listado 2-56: Implementando el tipo `checkbox`

El tipo **checkbox** es similar al tipo **radio**, pero tenemos que asignar diferentes nombres a cada elemento porque el usuario puede seleccionar varias opciones al mismo tiempo. Cuando el usuario selecciona una o más opciones, los valores de todos esos elementos se envían al servidor. Esta clase de campo de entrada también puede incluir el atributo **checked** para seleccionar opciones por defecto. En nuestro ejemplo seleccionamos dos valores: 15 y 30.

Figura 2-34: Campos de entrada de tipo `checkbox`

Otro tipo de entrada que genera una herramienta visual es **date**. Este control le ayuda al usuario a seleccionar una fecha. Algunos navegadores lo implementan como un calendario que se muestra cada vez que el usuario hace clic en el campo. El valor enviado al servidor por este tipo de campos tiene la sintaxis año-mes-día, por lo que si queremos especificar un valor inicial o el navegador no facilita una herramienta para seleccionarlo, debemos declararlo en este formato.

```
<input type="date" name="fecha" value="2017-02-05">
```

***Listado 2-57:** Implementando el tipo* **date**

El elemento **<input>** en el Listado 2-57 crea un campo de entrada con la fecha 2017-02-05 como valor por defecto. Si este elemento se muestra en un navegador que facilita un calendario para seleccionar la fecha, como Google Chrome, la fecha inicial seleccionada en el calendario será la que defina el atributo **value**.

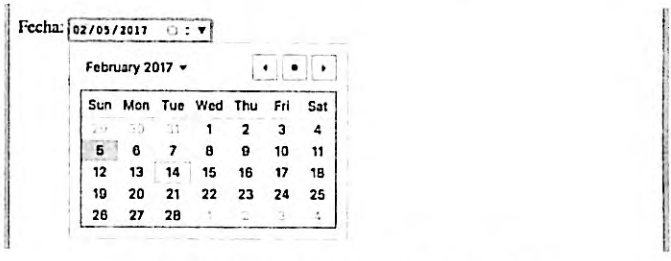

***Figura 2-35:** Campo de entrada de tipo* **date**

El tipo **date** no es el único disponible para insertar fechas. HTML también ofrece el tipo **datetime-local** para seleccionar fecha y hora, el tipo **week** para seleccionar una semana, el tipo **month** para seleccionar un mes, y el tipo **time** para seleccionar horas y minutos. Estos tipos se crearon con diferentes propósitos y, por lo tanto, esperan valores con diferentes sintaxis. El tipo **datetime-local** espera un valor que representa la fecha y la hora en el formato año-mes-día horas:minutos:segundos, con la letra T separando la fecha de la hora, como en 2017-02-05T10:35:00. El tipo **week** espera un valor con la sintaxis **2017-W30**, donde **2017** es el año y **30** es el número de la semana. El tipo **month** espera una sintaxis año-mes. Y finalmente, el tipo **time** espera un valor que representa horas y minutos con la sintaxis horas:minutos (el carácter : es convertido a la cadena de caracteres %3ª cuando el valor se envía al servidor).

Hágalo usted mismo: reemplace los elementos **<input>** en su archivo HTML por el elemento del Listado 2-57 y abra el documento en su navegador. Haga clic en el elemento. En un navegador moderno, debería ver una ventana con un calendario donde puede seleccionar una fecha. Cambie el tipo de entrada en el elemento **<input>** por cualquiera de los tipos mencionados arriba para ver las clases de herramientas que facilita el navegador para introducir estos tipos de valores. Si pulsa el botón **Enviar**, el valor seleccionado o introducido en el campo se agrega a la URL y el navegador intenta acceder al archivo procesar.php en el servidor para procesarlo. Debido a que este archivo aún no existe, el navegador devuelve

un error, pero al menos podrá ver en la barra de navegación cómo se incluyen los valores en la URL. Más adelante en este capítulo estudiaremos el modo de enviar un formulario y cómo procesar los valores en el servidor.

Además de los tipos de campos de entrada disponibles para introducir fechas, existe un tipo de campo llamado **color** que ofrece una interfaz predefinida para seleccionar un valor. Generalmente, el valor esperado por estos campos es un número hexadecimal, como #00FF00.

```
<input type="color" name="micolor" value="#99BB00">
```

***Listado 2-58:** Implementando el tipo* `color`

Un elemento **<input>** de tipo **color** se presenta en la pantalla como un rectángulo pintado del color determinado por defecto, seleccionado por el usuario, o definido por el atributo **value**. Cuando el usuario hace clic en este rectángulo, el navegador abre una herramienta que nos permite seleccionar un nuevo color.

***Figura 2-36:** Campo de entrada de tipo* `color`

Lo básico: los colores se pueden expresar en varios formatos y seleccionar desde sistemas de colores diferentes. La nomenclatura más común es la hexadecimal, cuya sintaxis incluye el carácter # seguido por tres valores hexadecimales que representan los componentes del color (rojo, verde, y azul). Estudiaremos cómo definir colores en el Capítulo 3.

Hasta el momento hemos usado un elemento **<input>** de tipo **submit** para crear el botón que el usuario tiene que pulsar para enviar el formulario. Este tipo de entrada crea un botón estándar identificado con un título. Si queremos mejorar el diseño, podemos implementar el tipo de entrada **image** que crea un botón con una imagen. Estos tipos de campos requieren el atributo **src** con la URL del archivo que contiene la imagen que queremos usar para el botón, y pueden incluir los atributos **width** y **height** para definir su tamaño.

```
<form name="formulario" method="get" action="procesar.php">
  <p><label>Name: <input type="text" name="nombre"></label></p>
  <p><label>Age: <input type="text" name="edad"></label></p>
  <p><input type="image" src="botonenviar.png" width="100"></p>
</form>
```

***Listado 2-59:** Implementando el tipo* `image` *para crear un botón personalizado*

El formulario del Listado 2-59 crea el botón para enviar el formulario con un elemento **<input>** de tipo **image**. El elemento carga la imagen del archivo botonenviar.png y la

muestra en la pantalla. Cuando el usuario hace clic en la imagen, el formulario se envía del mismo modo que con los botones que hemos usado anteriormente.

Figura 2-37: Campo de entrada de tipo `image`

Hágalo usted mismo: reemplace el formulario en su archivo HTML por el formulario del Listado 2-59. Descargue la imagen botonenviar.png desde nuestro sitio web. Abra el documento en su navegador. La imagen tiene un tamaño de 150 píxeles, pero como declaramos el atributo `width` con un valor de 100, el navegador reduce el ancho de la imagen a 100 píxeles. Elimine este atributo para ver la imagen en sus dimensiones originales.

Aunque los botones creados con elementos `<input>` son probablemente más que suficientes para la mayoría de los proyectos, HTML ofrece un elemento más versátil para crear botones llamado `<button>`. Este elemento incluye el atributo `type` para determinar el tipo de botón que queremos generar. Por ejemplo, si queremos crear un botón para enviar el formulario, debemos declarar el valor `submit`.

```
<form name="formulario" method="get" action="procesar.php">
  <p><label>Nombre: <input type="text" name="nombre"></label></p>
  <p><label>Edad: <input type="text" name="edad"></label></p>
  <p><button type="submit">Enviar Formulario</button></p>
</form>
```

Listado 2-60: Implementando el elemento `<button>` *para crear un botón*

Lo básico: el elemento `<button>` crea un botón estándar con las mismas características que el botón creado por el elemento `<input>`. La diferencia es que el título de estos botones se define entre las etiquetas de apertura y cierre, lo cual nos permite usar otros elementos HTML e incluso imágenes para declararlo. Por esta razón, el elemento `<button>` es el que se prefiere cuando queremos personalizarlo usando estilos CSS o cuando queremos usarlo para ejecutar códigos JavaScript, como veremos en próximos capítulos.

El elemento `<input>` permite al usuario insertar o seleccionar varios tipos de valores, pero los campos de entrada generados por este elemento nos dejan introducir una sola línea de texto. HTML ofrece el elemento `<textarea>` para insertar múltiples líneas de texto. El elemento está compuesto por etiquetas de apertura y cierre, y puede incluir los atributos `rows` y `cols` para definir el ancho y la altura del área en caracteres.

```
<form name="formulario" method="get" action="procesar.php">
  <p><label>Texto: <textarea name="texto" cols="50"
rows="6"></textarea></label></p>
  <p><input type="submit" value="Enviar"></p>
</form>
```

Listado 2-61: Implementando el elemento `<textarea>`

En la ventana del navegador, el elemento **`<textarea>`** se representa por un recuadro vacío del tamaño determinado por sus atributos o los estilos por defecto. El texto insertado queda limitado por el ancho del área, pero se puede extender verticalmente todo lo que sea necesario. Si el área no es lo suficientemente larga para contener el texto completo, el navegador muestra barras laterales con las que el usuario puede desplazar el contenido.

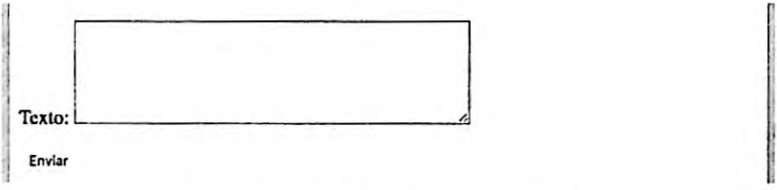

Figura 2-38: El elemento `<textarea>`

Hágalo usted mismo: reemplace el formulario en su archivo HTML por el formulario del Listado 2-61 y abra el documento en su navegador. Escriba varias líneas de texto para ver cómo se distribuye el texto dentro del área.

Lo básico: si necesita declarar un valor inicial para el elemento **`<textarea>`**, puede insertarlo entre las etiquetas de apertura y cierre.

Además de los tipos de campos de entrada **`radio`** y **`checkbox`** estudiados anteriormente, HTML ofrece el elemento **`<select>`** para presentar una lista de valores al usuario. Cuando el usuario hace clic en este elemento, la lista se muestra en una ventana desplegable, y luego el valor seleccionado por el usuario se inserta en el campo. Debido a que el elemento **`<select>`** no genera un campo de entrada, el usuario no puede insertar valores distintos de los incluidos en la lista.

El elemento **`<select>`** trabaja junto con el elemento **`<option>`** para definir las opciones. El elemento **`<select>`** debe incluir el atributo **`name`** para identificar el valor, y cada elemento **`<option>`** debe incluir el atributo **`value`** para definir el valor que representa.

```
<form name="formulario" method="get" action="procesar.php">
  <p>
    <label for="listado">Libros: </label>
    <select name="libro" id="listado">
      <option value="1">IT</option>
      <option value="2">Carrie</option>
      <option value="3">El Resplandor</option>
      <option value="4">Misery</option>
    </select>
  </p>
```

```
  <p><input type="submit" value="Enviar"></p>
</form>
```

Listado 2-62: Implementando el elemento `<select>`

En el ejemplo del Listado 2-62, incluimos cuatro opciones, una para cada libro que queremos mostrar en la lista. Los valores de las opciones son definidos desde 1 a 4. Por consiguiente, cada vez que el usuario selecciona un libro y envía el formulario, el número correspondiente a ese libro se envía al servidor con el identificador "libro" (el valor del atributo **name**).

Figura 2-39: Seleccionando un valor con el elemento `<select>`

Otra manera de crear una lista predefinida es con el elemento **<datalist>**. Este elemento define una lista de ítems que, con la ayuda del atributo **list**, se puede usar como sugerencia en un campo de entrada. Al igual que las opciones para el elemento **<select>**, las opciones para este elemento se definen por elementos **<option>**, pero en este caso deben incluir el atributo **label** con una descripción del valor. El siguiente ejemplo crea un formulario con un campo de entrada para insertar un número telefónico. El código incluye un elemento **<datalist>** con dos elementos **<option>** que definen los valores que queremos sugerir al usuario.

```
<form name="formulario" method="get" action="procesar.php">
  <datalist id="datos">
    <option value="123123123" label="Teléfono 1">
    <option value="456456456" label="Teléfono 2">
  </datalist>
  <p><label>Teléfono: <input type="tel" name="telefono"
list="datos"></label></p>
  <p><input type="submit" value="Enviar"></p>
</form>
```

Listado 2-63: Sugiriendo una lista de valores con el elemento `<datalist>`

Para conectar un campo de entrada con un elemento **<datalist>**, tenemos que incluir el atributo **list** en el elemento **<input>** con el mismo valor que usamos para identificar el elemento **<datalist>**. Para este propósito, en el formulario del Listado 2-63 incluimos el atributo **id** en el elemento **<datalist>** con el valor "datos" y luego asignamos este mismo valor al atributo **list** del elemento **<input>**. En consecuencia, el campo de entrada muestra una flecha que despliega una lista de valores predefinidos que el usuario puede seleccionar para completar el formulario.

Figura 2-40: Valores predefinidos con el elemento `<datalist>`

Hágalo usted mismo: reemplace el formulario en su archivo HTML con el formulario del Listado 2-63. Abra el documento en su navegador y haga clic en la flecha del lado derecho del campo. Debería ver algo similar a lo que se muestra en la Figura 2-40.

HTML incluye otros dos elementos que podemos usar en un formulario: **<progress>** y **<meter>**, que no se consideran elementos de formulario, pero debido a que representan medidas, son realmente útiles cuando nuestro formulario produce esta clase de información.

El elemento **<progress>** se usa para informar del progreso en la ejecución de una tarea. Requiere dos atributos que determinan el valor actual y el máximo. El atributo **value** indica el progreso logrado hasta el momento, y el atributo **max** declara el valor que necesitamos alcanzar para dar por finalizada la tarea.

```
<progress value="30" max="100">0%</progress>
```

***Listado 2-64:** Implementando el elemento* `<progress>`

Los navegadores representan este elemento con una barra de dos colores. Por defecto, la porción de la barra que representa el progreso se muestra en color azul. Si el navegador no reconoce el elemento, el valor entre las etiquetas de apertura y cierre se muestra en su lugar.

***Figura 2-41:** Barra de progreso*

Al igual que el elemento **<progress>**, el elemento **<meter>** se usa para mostrar una escala, pero no representa progreso. Su propósito es representar un rango de valores predefinidos (por ejemplo, el espacio ocupado en un disco duro). Este elemento cuenta con varios atributos asociados. Los atributos **min** y **max** determinan los límites del rango, **value** determina el valor medido, y **low**, **high** y **optimum** se usan para segmentar el rango en secciones diferenciadas y declarar la posición óptima.

```
<meter value="60" min="0" max="100" low="40" high="80"
optimum="100">60</meter>
```

***Listado 2-65:** Implementando el elemento* `<meter>`

El código del Listado 2-65 genera una barra en la pantalla que muestra un nivel de 60 en una escala de 0 a 100 (de acuerdo con los valores declarados por los atributos **value**, **min** y **max**). El color de la barra generada por el elemento depende de los niveles determinados por los atributos **low**, **high** y **optimum**. Debido a que el valor actual en nuestro ejemplo se encuentra entre los valores de los atributos **low** y **high**, la barra se muestra en color amarillo.

***Figura 2-42:** Barra creada con el elemento* `<meter>`

Enviando el formulario

Al enviar un formulario, los datos introducidos se envían al servidor usando la sintaxis nombre/valor para cada elemento, donde *nombre* es el valor asignado al atributo `name` del elemento y *valor* es el valor que introduce el usuario. Por ejemplo, si insertamos el texto "Roberto" en un campo de entrada con el nombre "minombre", el par nombre/valor que se envía al servidor será minombre/Roberto.

Los navegadores usan dos métodos para enviar esta información: GET y POST. El método se declara asignando los valores `GET` o `POST` al atributo `method` del elemento `<form>`, como hemos hecho en ejemplos anteriores, aunque el método a usar depende del tipo de información gestionada por el formulario. Como mencionamos en el Capítulo 1, los navegadores se comunican con el servidor usando un protocolo llamado HTTP. Cuando el navegador quiere acceder a un documento, envía una solicitud HTTP al servidor. Una solicitud HTTP es un mensaje que le dice al servidor cuál es el recurso al que el navegador quiere acceder y lo que quiere hacer con el mismo (descargar el documento, procesar información, etc.). El mensaje está compuesto por la URL del recurso, los datos asociados con la solicitud, como la fecha y el idioma, y un cuerpo con información adicional. Los pares nombre/valor producidos por un formulario se envían al servidor dentro de estas solicitudes HTTP. Si el método se ha declarado como `GET`, los pares nombre/valor se agregan al final de la URL, pero si el método se ha declarado como `POST`, los valores se incluyen en el cuerpo de la solicitud. Esto significa que la información enviada con el método GET es visible para el usuario (el usuario puede ver la URL con todos los pares nombre/valor en la barra de navegación del navegador), pero la información enviada con el método POST se oculta dentro de la solicitud. En consecuencia, si la información es sensible o privada, debemos usar el método POST, pero si la información no es sensible, como valores de búsqueda insertados por el usuario, podemos usar el método GET.

Así mismo, tenemos que considerar que el método POST se puede usar para enviar una cantidad ilimitada de información, pero el método GET tiene que adaptarse a las limitaciones presentadas por las URL. Esto se debe a que el largo de una URL es limitado. Si la información insertada en el formulario es muy extensa, se podría perder.

El siguiente ejemplo presenta un formulario con un único campo de entrada para ilustrar cómo funciona este proceso. El elemento `<input>` se identifica con el nombre "val", y el método utilizado para enviar la información se declara como GET, lo que significa que el valor insertado por el usuario se agregará a la URL.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>Formularios</title>
</head>
<body>
  <section>
    <form name="formulario" method="get" action="procesar.php">
      <p><input type="text" name="val"></p>
      <p><input type="submit" value="Enviar"></p>
    </form>
  </section>
</body>
</html>
```

***Listado 2-66:** Formularios con el método GET*

En el ejemplo del Listado 2-66, un archivo llamado procesar.php se declara a cargo de procesar la información. Cuando el usuario pulsa el botón **Enviar**, el navegador crea una solicitud HTTP que incluye la URL que apunta a este archivo. Debido a que el método del formulario se ha declarado como GET, el nombre del campo de entrada y el valor insertado por el usuario se agregan a la URL, tal como se muestra a continuación.

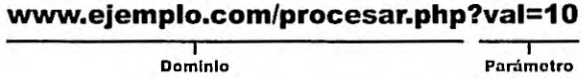

Figura 2-43: Par nombre/valor en la URL

Cuando la información se envía con el método GET, los pares nombre/valor se agregan al final de la URL separados por el carácter =, y el primer par es precedido por el carácter ?. Si existe más de un par de valores, los restantes se agregan a la URL separados por el carácter &, como en www.ejemplo.com/procesar.php?val1=10&val2=20.

Al otro lado, el servidor recibe esta solicitud, lee la URL, extrae los valores y ejecuta el código en el archivo procesar.php. Este código debe procesar la información recibida y producir una respuesta. La forma en la que se realiza esta tarea depende del lenguaje de programación que utilizamos y lo que queremos lograr. Por ejemplo, para leer los valores enviados con el método GET, PHP los ofrece en un listado llamado `$_GET`. La sintaxis requerida para obtener el valor incluye su nombre entre corchetes.

```
<?php
  print('El valor es: '.$_GET['val']);
?>
```

***Listado 2-67:** Procesando los datos en el servidor con PHP (procesar.php)*

El ejemplo del Listado 2-67 muestra cómo procesar valores con PHP. Las etiquetas **<?php** y **?>** indican al servidor que este es código PHP y que tiene que ser ejecutado como tal. El código puede ser extenso o estar compuesto por solo unas pocas instrucciones, dependiendo de lo que necesitamos. Nuestro ejemplo incluye una sola instrucción para ilustrar cómo se procesa la información. Esta instrucción, llamada `print()`, toma los valores entre paréntesis y los incluye en un archivo que se va a devolver al navegador como respuesta. En este caso, agregamos el valor recibido por medio de la solicitud al texto "El valor es: ". Si enviamos el valor 10, como en el ejemplo de la Figura 2-43, el servidor genera un archivo con el texto "El valor es: 10" y lo envía de regreso al navegador.

Lo básico: el código PHP de nuestro ejemplo utiliza el nombre `$_GET` para capturar la información recibida desde el navegador porque el método del formulario se ha declarado como GET, pero si cambiamos el método a POST, debemos utilizar el nombre `$_POST`.

El archivo producido por el servidor a través de un código PHP es un archivo HTML. Por propósitos didácticos, no incluimos ningún elemento HTML en el ejemplo del Listado 2-67, pero siempre deberíamos generar un documento HTML válido en respuesta. Existen varias maneras de definir un documento en PHP. La más simple es crear el documento HTML como lo hemos hecho anteriormente pero dentro de un archivo PHP, e incluir el código PHP donde queremos mostrar el resultado. Por ejemplo, el siguiente ejemplo inserta código PHP dentro de un elemento **<p>**.

Cuando el servidor abre el archivo procesar.php con este documento, ejecuta el código PHP, inserta el resultado dentro de las etiquetas <p>, y devuelve el archivo al servidor. El resultado es el mismo que si hubiéramos creado un documento estático con el texto "El valor es: 10", pero el texto se genera dinámicamente en el servidor con los valores recibidos desde el formulario.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>Respuesta</title>
</head>
<body>
  <section>
    <p>
      <?php
        print('El valor es: '.$_GET['val']);
      ?>
    </p>
  </section>
</body>
</html>
```

Listado 2-68: Generando un documento HTML que incluye código PHP (procesar.php)

Hágalo usted mismo: para probar este ejemplo, tiene que subir los archivos a un servidor que pueda ejecutar código PHP. Si no posee una cuenta de alojamiento con estas características, puede instalar un servidor en su ordenador con paquetes como MAMP, introducido en el Capítulo 1 (si no sabe cómo usar un servidor, no se preocupe, este procedimiento no es necesario para realizar la mayoría de los ejemplos de este libro). Cree un nuevo archivo HTML con el código del Listado 2-66, y un archivo llamado procesar.php con el código del Listado 2-67 o 2-68. Suba los archivos al servidor y abra el documento en su navegador. Inserte un valor dentro del campo de entrada y pulse el botón Enviar. El navegador debería mostrar una nueva página que contenga el texto que comienza con la cadena de caracteres "El valor es: " y termina con el valor que ha insertado en el formulario.

IMPORTANTE: este es simplemente un ejemplo de cómo se procesa la información y cómo el navegador transmite los datos introducidos en el formulario al servidor. Estudiaremos otros ejemplos parecidos en próximos capítulos, pero no es el propósito de este libro enseñar cómo programar en PHP u otro lenguaje de programación de servidor. Para aprender más sobre PHP, vaya a nuestro sitio web y visite las secciones Enlaces y Vídeos.

Atributos globales

HTML define atributos globales que son exclusivos de elementos de formulario. Los siguientes son los más utilizados.

disabled—Este es un atributo booleano que desactiva el elemento. Cuando el atributo está presente, el usuario no puede introducir valores o interactuar con el elemento.

readonly—Este atributo indica que el valor del elemento no se puede modificar.

placeholder—Este atributo muestra un texto en el fondo del elemento que indica al usuario el valor que debe introducir.

autocomplete—Este atributo activa o desactiva la función de autocompletar. Los valores disponibles son **on** y **off**.

novalidate—Este es un atributo booleano para el elemento **<form>** que indica que el formulario no debería ser validado.

formnovalidate—Este es un atributo booleano para los elementos **<button>** e **<input>** de tipo **submit** e **image** que indica que el formulario al que pertenecen no debería ser validado.

required—Este es un atributo booleano que indica al navegador que el usuario debe seleccionar o insertar un valor en el elemento para validar el formulario.

multiple—Este es un atributo booleano que indica al navegador que se pueden insertar múltiples valores en el campo (se aplica a elementos **<input>** de tipo **email** y **file**).

autofocus—Este atributo booleano solicita al navegador que mueva el foco al elemento tan pronto como se carga el documento.

pattern—Este atributo define una expresión regular que el navegador debe usar para validar el valor insertado en el campo.

form—Este atributo asocia el elemento con un formulario. Se usa para conectar un elemento con un formulario cuando el elemento no se define entre las etiquetas **<form>**. El valor asignado a este atributo debe ser el mismo asignado al atributo **id** del elemento **<form>**.

spellcheck—Este atributo solicita al navegador que compruebe la ortografía y gramática del valor introducido en el campo. Los valores disponibles son **true** (verdadero) y **false** (falso).

Los atributos **disabled** y **readonly** tienen un propósito similar, no permitir al usuario interactuar con el elemento, pero se aplican en diferentes circunstancias. El atributo **disabled** normalmente se implementa cuando queremos mostrar al usuario que el elemento puede estar disponible en otras condiciones, como cuando el control no es aplicable en el país del usuario, por ejemplo. Por otro lado, el atributo **readonly** se implementa cuando solo existe un valor posible y no queremos que el usuario lo cambie. Por ejemplo, en el siguiente formulario, el usuario no puede introducir la edad.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>Formularios</title>
</head>
<body>
  <section>
    <form name="formulario" method="get" action="procesar.php">
      <p><label>Nombre: <input type="text" name="nombre"></label></p>
```

```
      <p><label>Edad: <input type="text" name="edad" disabled></label></p>
      <p><input type="submit" value="Enviar"></p>
    </form>
  </section>
</body>
</html>
```

Listado 2-69: Implementando el atributo `disabled`

Los elementos afectados por los atributos **disabled** y **readonly** se muestran en colores más claros para advertir al usuario de que no son controles normales. Otro atributo que afecta la apariencia de un elemento es **placeholder**. Este se usa en campos de entrada para ofrecer una pista (una palabra o frase) que ayude al usuario a introducir el valor correcto. El siguiente ejemplo inserta esta ayuda en un campo de búsqueda.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>Formularios</title>
</head>
<body>
  <section>
    <form name="formulario" method="get" action="procesar.php">
      <p><label>Buscar: <input type="search" name="buscar"
placeholder="Término a buscar"></label></p>
      <p><input type="submit" value="Buscar"></p>
    </form>
  </section>
</body>
</html>
```

Listado 2-70: Implementando el atributo `placeholder`

El valor de este atributo lo muestran los navegadores dentro del campo hasta que el usuario inserta un valor.

Figura 2-44: Campo de entrada con un mensaje de ayuda

Una de las características de los formularios es que tienen la capacidad de validar los datos introducidos. Por defecto, los formularios validan los datos a menos que el atributo **novalidate** sea declarado. Este atributo booleano es específico de elementos **<form>**. Cuando es incluido, el formulario se envía sin validar. La presencia de este atributo afecta al formulario de forma permanente, pero a veces el proceso de validación es requerido solo en ciertas circunstancias. Por ejemplo, cuando la información insertada debe ser grabada para permitir al usuario continuar con el trabajo más adelante. En casos como este, podemos

implementar el atributo **formnovalidate**. Este atributo está disponible para los elementos que crean los botones para enviar el formulario. Cuando los datos se envían con un botón que contiene este atributo, el formulario no es validado.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>Formularios</title>
</head>
<body>
  <section>
    <form name="formulario" method="get" action="procesar.php">
      <p><label>Correo: <input type="email" name="correo"></label></p>
      <p>
        <input type="submit" value="Enviar">
        <input type="submit" value="Grabar" formnovalidate>
      </p>
    </form>
  </section>
</body>
</html>
```

***Listado 2-71:** Enviando un formulario sin validar con el atributo* `formnovalidate`

En el ejemplo del Listado 2-71, el formulario será validado en circunstancias normales, pero incluimos un segundo botón con el atributo **formnovalidate** para poder enviar el formulario sin pasar por el proceso de validación. El botón **Enviar** requiere que el usuario introduzca una cuenta de correo válida, pero el botón **Grabar** no incluye este requisito.

Cuando usamos el tipo **email** para recibir una cuenta de correo, como en el ejemplo anterior, el navegador controla si el valor introducido es una cuenta de correo, pero valida el campo cuando se encuentra vacío. Esto se debe a que el campo no es obligatorio. HTML ofrece el atributo **required** para cambiar esta condición. Cuando se incluye el atributo **required**, el campo solo será válido si el usuario introduce un valor y este valor cumple con los requisitos de su tipo. El siguiente ejemplo implementa este atributo para forzar al usuario a introducir un correo electrónico.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>Formularios</title>
</head>
<body>
  <section>
    <form name="formulario" method="get" action="procesar.php">
      <p><label>Correo: <input type="email" name="correo"
required></label></p>
      <p>
        <input type="submit" value="Enviar">
        <input type="submit" value="Grabar" formnovalidate>
      </p>
```

```
    </form>
  </section>
</body>
</html>
```

Listado 2-72: Declarando una entrada `email` *como campo requerido*

Hágalo usted mismo: cree un archivo HTML con el código del Listado 2-72. Abra el documento en su navegador y pulse el botón enviar. Debería recibir un mensaje de error en el campo **correo** indicando que debe insertar un valor. Pulse el botón **Grabar**. Como este botón incluye el atributo **formnovalidate**, el error ya no se muestra y se envía el formulario.

Otro atributo que se usa para validación es **pattern**. Algunos tipos de campos de entrada validan cadenas de caracteres específicas, pero no pueden hacer nada cuando el valor no es estándar, como en el caso de los códigos postales. No existe un tipo de campo predeterminado para esta clase de valores. El atributo **pattern** nos permite crear un filtro personalizado usando expresiones regulares.

Las expresiones regulares son textos compuestos por una serie de caracteres que definen un patrón de concordancia. Por ejemplo, los caracteres 0-9 determinan que solo se aceptan los números entre 0 y 9. Utilizando esta clase de expresiones, podemos crear un filtro personalizado para validar cualquier valor que necesitemos. El siguiente formulario incluye un campo de entrada que solo acepta números con cinco dígitos.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>Formularios</title>
</head>
<body>
  <section>
    <form name="formulario" method="get" action="procesar.php">
      <p><label>Código Postal: <input pattern="[0-9]{5}" name="cp"
title="Inserte su código postal"></label></p>
      <p><input type="submit" value="Enviar"></p>
    </form>
  </section>
</body>
</html>
```

Listado 2-73: Personalizando campos de entrada con el atributo `pattern`

El elemento **<input>** en el Listado 2-73 incluye el atributo **pattern** con el valor **[0-9]{5}**. Esta expresión regular determina que el valor debe tener exactamente cinco caracteres y que esos caracteres deben ser números entre 0 y 9. En consecuencia, solo podemos insertar números de cinco dígitos; cualquier otro carácter o tamaño devolverá un error.

Estos tipos de entrada también pueden incluir el atributo **title** para explicar al usuario cuál es el valor esperado. Este mensaje complementa el mensaje de error estándar que muestra el navegador, tal como ilustra la Figura 2-45.

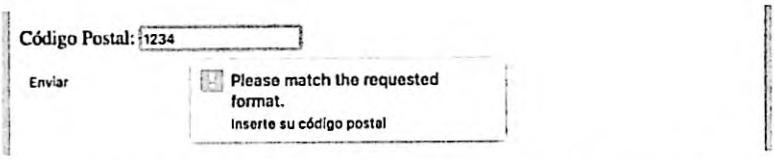

Figura 2-45: Error en un campo de entrada con un patrón personalizado

IMPORTANTE: no es el propósito de este libro enseñar cómo trabajar con expresiones regulares. Para más información, visite nuestro sitio web y siga los enlaces de este capítulo.

Los tipos de entrada **email** y **file** solo permiten al usuario introducir un valor a la vez. Si queremos permitir la inserción de múltiples valores, tenemos que incluir el atributo **multiple**. Con este atributo, el usuario puede insertar todos los valores que quiera separados por coma. El siguiente ejemplo incluye un campo de entrada que acepta múltiples direcciones de correo.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>Formularios</title>
</head>
<body>
  <section>
    <form name="formulario" method="get" action="procesar.php">
      <p><label>Correo: <input type="email" name="correo"
multiple></label></p>
      <p><input type="submit" value="Enviar"></p>
    </form>
  </section>
</body>
</html>
```

Listado 2-74: Declarando un campo de entrada `email` *como campo múltiple*

Además de los atributos de validación, existen otros atributos que pueden ayudar al usuario a decidir qué valores introducir. Por ejemplo, el atributo **autocomplete** activa una herramienta que le sugiere al usuario qué introducir según los valores insertados previamente. Los valores disponibles para este atributo son **on** y **off** (activar y desactivar la herramienta). Este atributo también se puede implementar en el elemento **<form>** para que afecte a todos los elementos del formulario. El siguiente formulario desactiva las sugerencias para una búsqueda.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>Formularios</title>
</head>
<body>
```

```
  <section>
    <form name="formulario" method="get" action="procesar.php">
      <p><label>Buscar: <input type="search" name="buscar"
autocomplete="off"></label></p>
      <p><input type="submit" value="Buscar"></p>
    </form>
  </section>
</body>
</html>
```

Listado 2-75: Implementando el atributo `autocomplete`

Hágalo usted mismo: cree un archivo HTML con el código del Listado 2-75. Abra el documento en su navegador. Inserte un valor y presione el botón **Enviar**. Repita el proceso. El navegador no debería sugerir ningún valor a insertar. Cambie el valor del atributo **autocomplete** a **on** y repita el proceso. En esta oportunidad, cada vez que introduzca un valor, el navegador le mostrará un listado con los valores insertados previamente que comienzan con los mismos caracteres.

Otro atributo que puede ayudar al usuario a decidir qué insertar es **autofocus**. En este caso, el atributo establece el foco en el elemento cuando se carga el documento, sugiriendo al usuario qué valor insertar primero. El siguiente ejemplo incluye dos campos de entrada para insertar el nombre y la edad del usuario, pero el campo para la edad incluye el atributo **autofocus** y, por lo tanto, el navegador posicionará el cursor en este campo por defecto.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>Formularios</title>
</head>
<body>
  <section>
    <form name="formulario" method="get" action="procesar.php">
      <p><label>Nombre: <input type="text" name="nombre"></label></p>
      <p><label>Edad: <input type="text" name="edad" autofocus></label></p>
      <p><input type="submit" value="Enviar"></p>
    </form>
  </section>
</body>
</html>
```

Listado 2-76: Implementando el atributo `autofocus`

Otra herramienta que pueden activar automáticamente los navegadores es el control de ortografía. A pesar de la utilidad de esta herramienta y de que los usuarios esperan casi siempre tenerla a su disposición, puede resultar inapropiada en ciertas circunstancias. La herramienta se activa por defecto, pero podemos incluir el atributo **spellcheck** con el valor **false** para desactivarla, como hacemos en el siguiente ejemplo.

```
<!DOCTYPE html>
<html lang="es">
```

```
<head>
  <meta charset="utf-8">
  <title>Formularios</title>
</head>
<body>
  <section>
    <form name="formulario" method="get" action="procesar.php">
      <p><label>Texto: <textarea name="texto" cols="50" rows="6"
spellcheck="false"></textarea></label></p>
      <p><input type="submit" value="Enviar"></p>
    </form>
  </section>
</body>
</html>
```

Listado 2-77: Desactivando el control de ortografía

Finalmente, existe otro atributo útil que podemos implementar para declarar como parte del formulario elementos que no se han incluido entre las etiquetas **<form>**. Por ejemplo, si tenemos un campo de entrada en el área de navegación, pero queremos que el elemento se envíe con el formulario definido en el área central del documento, podemos conectar este elemento con el formulario usando el atributo **form** (el valor de este atributo debe coincidir con el valor del atributo **id** asignado al elemento **<form>**).

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>Formularios</title>
</head>
<body>
  <nav>
    <p><input type="search" name="buscar" form="formulario"></p>
  </nav>
  <section>
    <form name="formulario" id="formulario" method="get"
action="procesar.php">
      <p><input type="submit" value="Enviar"></p>
    </form>
  </section>
</body>
</html>
```

Listado 2-78: Declarando elementos de formulario en una parte diferente del documento

IMPORTANTE: los formularios son extremadamente útiles en el desarrollo web, y se requieren en la mayoría de los sitios y aplicaciones web modernas. Los ejemplos de este capítulo se han creado con propósitos didácticos y se han simplificado al máximo para que pueda probar los elementos y controles disponibles. En próximos capítulos, estudiaremos cómo implementar formularios en situaciones más prácticas y expandirlos con CSS y JavaScript.

Capítulo 3
CSS

3.1 Estilos

En el capítulo anterior, hemos aprendido a crear un documento HTML, a organizar su estructura, y a determinar qué elementos son más apropiados para representar su contenido. Esta información nos permite definir el documento, pero no determina cómo se mostrará en pantalla. Desde la introducción de HTML5, esa tarea es responsabilidad de CSS. CSS es un lenguaje que facilita instrucciones que podemos usar para asignar estilos a los elementos HTML, como colores, tipos de letra, tamaños, etc. Los estilos se deben definir con CSS y luego asignar a los elementos hasta que logramos el diseño visual que queremos para nuestra página.

Por razones de compatibilidad, los navegadores asignan estilos por defecto a algunos elementos HTML. Esta es la razón por la que en el Capítulo 2 algunos elementos tenían márgenes o generaban saltos de línea, pero otros eran definidos de forma parecida (tenían el mismo tipo de letra y colores, por ejemplo). Algunos de estos estilos son útiles, pero la mayoría deben ser reemplazados o complementados con estilos personalizados. En CSS, los estilos personalizados se declaran con propiedades. Un estilo se define declarando el nombre de la propiedad y su valor separados por dos puntos. Por ejemplo, el siguiente código declara una propiedad que cambia el tamaño de la letra a 24 píxeles (debido a que algunas propiedades pueden incluir múltiples valores separados por un espacio, debemos indicar el final de la línea con un punto y coma).

```
font-size: 24px;
```

***Listado 3-1:** Declarando propiedades CSS*

Si la propiedad del Listado 3-1 se aplica a un elemento, el texto contenido por ese elemento se mostrará en la pantalla con el tipo de letra definido por defecto, pero con un tamaño de 24 píxeles.

Lo básico: aunque los píxeles (`px`) son las unidades de medida que más se implementan, más adelante veremos que en algunas circunstancias, especialmente cuando creamos nuestro sitio web con diseño web adaptable, pueden ser más apropiadas otras unidades. Las unidades más utilizadas son `px` (píxeles), `pt` (puntos), `in` (pulgadas), `cm` (centímetros), `mm` (milímetros), `em` (relativo al tamaño de letra del elemento), `rem` (relativo al tamaño de letra del documento), y `%` (relativo al tamaño del contenedor del elemento).

La mayoría de las propiedades CSS pueden modificar un solo aspecto del elemento (el tamaño de la letra en este caso). Si queremos cambiar varios estilos al mismo tiempo, tenemos que declarar múltiples propiedades. CSS define una sintaxis que simplifica el proceso de asignar múltiples propiedades a un elemento. La construcción se llama *regla*. Una regla es una lista de propiedades declaradas entre llaves e identificadas por un selector. El selector indica qué elementos se verán afectados por las propiedades. Por ejemplo, la siguiente regla

se identifica con el selector `p` y, por lo tanto, sus propiedades se aplicarán a todos los elementos `<p>` del documento.

```
p {
  color: #FF0000;
  font-size: 24px;
}
```

Listado 3-2: Declarando reglas CSS

La regla del Listado 3-2 incluye dos propiedades con sus respectivos valores agrupadas por llaves (`color` y `font-size`). Si aplicamos esta regla a nuestro documento, el texto dentro de cada elemento `<p>` se mostrará en color rojo y con un tamaño de 24 píxeles.

Aplicando estilos

Las propiedades y las reglas definen los estilos que queremos asignar a uno o más elementos, pero estos estilos no se aplican hasta que los incluimos en el documento. Existen tres técnicas disponibles para este propósito. Podemos usar estilos en línea, estilos incrustados u hojas de estilo. El primero, estilos en línea, utiliza un atributo global llamado `style` para insertar los estilos directamente en el elemento. Este atributo está disponible en cada uno de los elementos HTML y puede recibir una propiedad o una lista de propiedades que se aplicarán al elemento al que pertenece. Si queremos asignar estilos usando esta técnica, todo lo que tenemos que hacer es declarar el atributo `style` en el elemento que queremos modificar y asignarle las propiedades CSS.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Este texto es el título del documento</title>
  <meta charset="utf-8">
</head>
<body>
  <main>
    <section>
      <p style="font-size: 20px;">Mi Texto</p>
    </section>
  </main>
</body>
</html>
```

Listado 3-3: Estilos en Línea

El documento del Listado 3-3 incluye un elemento `<p>` con el atributo `style` y el valor `font-size: 20px;`. Cuando el navegador lee este atributo, le asigna un tamaño de 20 píxeles al texto dentro del elemento `<p>`.

Hágalo usted mismo: cree un nuevo archivo HTML con el código del Listado 3-3 y abra el documento en su navegador. Debería ver el texto "Mi Texto" con letras en el tamaño definido por la propiedad `font-size`. Intente cambiar el valor de esta propiedad para ver cómo se presentan en pantalla los diferentes tamaños de letra.

Los estilos en línea son una manera práctica de probar estilos y ver cómo modifican los elementos, pero no se recomiendan para proyectos extensos. La razón es que el atributo **style** solo afecta al elemento en el que se ha declarado. Si queremos asignar el mismo estilo a otros elementos, tenemos que repetir el código en cada uno de ellos, lo cual incrementa innecesariamente el tamaño del documento, complicando su actualización y mantenimiento. Por ejemplo, si más adelante decidimos que en lugar de 20 píxeles el tamaño del texto en cada elemento **<p>** deber ser de 24 píxeles, tendríamos que cambiar cada estilo en cada uno de los elementos **<p>** del documento completo y en todos los documentos de nuestro sitio web.

Una alternativa es la de insertar las reglas CSS en la cabecera del documento usando selectores que determinan los elementos que se verán afectados. Para este propósito, HTML incluye el elemento **<style>**.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Este texto es el título del documento</title>
  <meta charset="utf-8">
  <style>
    p {
      font-size: 20px;
    }
  </style>
</head>
<body>
  <main>
    <section>
      <p>Mi texto</p>
    </section>
  </main>
</body>
</html>
```

Listado 3-4: Incluyendo estilos en la cabecera del documento

La propiedad declarada entre las etiquetas **<style>** en el documento del Listado 3-4 cumple la misma función que la declarada en el documento del Listado 3-3, pero en este ejemplo no tenemos que escribir el estilo dentro del elemento **<p>** que queremos modificar porque, debido al selector utilizado, todos se ven afectados por esta regla.

Declarar estilos en la cabecera del documento ahorra espacio y hace que el código sea más coherente y fácil de mantener, pero requiere que copiemos las mismas reglas en cada uno de los documentos de nuestro sitio web. Debido a que la mayoría de las páginas compartirán el mismo diseño, varias de estas reglas se duplicarán. La solución es mover los estilos a un archivo CSS y luego usar el elemento **<link>** para cargarlo desde cada uno de los documentos que lo requieran. Este tipo de archivos se denomina *hojas de estilo*, pero no son más que archivos de texto con la lista de reglas CSS que queremos asignar a los elementos del documento.

Como hemos visto en el Capítulo 2, el elemento **<link>** se usa para incorporar recursos externos al documento. Dependiendo del tipo de recurso que queremos cargar, tenemos que declarar diferentes atributos y valores. Para cargar hojas de estilo CSS, solo necesitamos los atributos **rel** y **href**. El atributo **rel** significa relación y especifica la relación entre el documento y el archivo que estamos incorporando, por lo que debemos declararlo con el valor **stylesheet**

para comunicarle al navegador que el recurso es un archivo CSS con los estilos requeridos para presentar la página. Por otro lado, el atributo **href** declara la URL del archivo a cargar.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Este texto es el título del documento</title>
  <meta charset="utf-8">
  <link rel="stylesheet" href="misestilos.css">
</head>
<body>
  <main>
    <section>
      <p>Mi texto</p>
    </section>
  </main>
</body>
</html>
```

***Listado 3-5:** Aplicando estilos CSS desde un archivo externo*

El documento del Listado 3-5 carga los estilos CSS desde el archivo misestilos.css. En este archivo tenemos que declarar las reglas CSS que queremos aplicar al documento, tal como lo hemos hecho anteriormente dentro del elemento **<style>**. El siguiente es el código que debemos insertar en el archivo misestilos.css para producir el mismo efecto logrado en ejemplos anteriores.

```
p {
  font-size: 20px;
}
```

***Listado 3-6:** Definiendo estilos CSS en un archivo externo*

La práctica de incluir estilos CSS en un archivo aparte es ampliamente utilizada por diseñadores y se recomienda para sitos web diseñados con HTML5, no solo porque podemos definir una sola hoja de estilo y luego incluirla en todos los documentos con el elemento **<link>**, sino porque podemos reemplazar todos los estilos a la vez simplemente cargando un archivo diferente, lo cual nos permite probar diferentes diseños y adaptar el sitio web a las pantallas de todos los dispositivos disponibles, tal como veremos en el Capítulo 5.

Hágalo usted mismo: cree un archivo HTML con el código del Listado 3-5 y un archivo llamado misestilos.css con el código del Listado 3-6. Abra el documento en su navegador. Debería ver el texto dentro del elemento **<p>** con un tamaño de 20 píxeles. Cambie el valor de la propiedad **font-size** en el archivo CSS y actualice la página para ver cómo se aplica el estilo al documento.

Hojas de estilo en cascada

Una característica importante del CSS es que los estilos se asignan en cascada (de ahí el nombre hojas de estilo en cascada o Cascading Style Sheets en inglés). Los elementos en los niveles bajos de la jerarquía heredan los estilos asignados a los elementos en los niveles más

altos. Por ejemplo, si asignamos la regla del Listado 3-6 a los elementos **<section>** en lugar de los elementos **<p>**, como se muestra a continuación, el texto en el elemento **<p>** de nuestro documento se mostrará con un tamaño de 20 píxeles debido a que este elemento es hijo del elemento **<section>** y, por lo tanto, hereda sus estilos.

```
section {
  font-size: 20px;
}
```

***Listado 3-7:** Heredando estilos*

Los estilos heredados de elementos en niveles superiores se pueden reemplazar por nuevos estilos definidos para los elementos en niveles inferiores de la jerarquía. Por ejemplo, podemos declarar una regla adicional para los elementos **<p>** que sobrescriba la propiedad **font-size** definida para el elemento **<section>** con un valor diferente.

```
section {
  font-size: 20px;
}
p {
  font-size: 36px;
}
```

***Listado 3-8:** Sobrescribiendo estilos*

Ahora, el elemento **<p>** se nuestro documento se muestra con un tamaño de 36 píxeles porque el valor de la propiedad **font-size** asignada a los elementos **<section>** se modifica con la nueva regla asignada a los elementos **<p>**.

***Figura 3-1:** Estilos en cascada*

Hágalo usted mismo: reemplace los estilos en su archivo misestilos.css por el código del Listado 3-8 y abra el documento del Listado 3-5 en su navegador. Debería ver el texto dentro del elemento **<p>** con un tamaño de 36 píxeles, tal como muestra la Figura 3-1.

3.2 Referencias

A veces es conveniente declarar todos los estilos en un archivo externo y luego cargar ese archivo desde cada documento que lo necesite, pero nos obliga a implementar diferentes mecanismos para establecer la relación entre las reglas CSS y los elementos dentro del documento que se verán afectados por las mismas.

Existen varios métodos para seleccionar los elementos que serán afectados por una regla CSS. En ejemplos anteriores hemos utilizado el nombre del elemento, pero también podemos

usar los valores de los atributos `id` y `class` para referenciar un solo elemento o un grupo de elementos, e incluso combinarlos para construir selectores más específicos.

Nombres

Una regla declarada con el nombre del elemento como selector afecta a todos los elementos de ese tipo encontrados en el documento. En ejemplos anteriores usamos el nombre `p` para modificar elementos **<p>**, pero podemos cambiar este nombre para trabajar con cualquier elemento en el documento que deseemos. Si en su lugar declaramos el nombre `span`, por ejemplo, se modificarán todos los textos dentro de elementos ****.

```
span {
  font-size: 20px;
}
```

***Listado 3-9:** Referenciando elementos* *por su nombre*

Si queremos asignar los mismos estilos a elementos con nombres diferentes, podemos declarar los nombres separados por una coma. En el siguiente ejemplo, la regla afecta a todos los elementos **<p>** y **** encontrados en el documento.

```
p, span {
  font-size: 20px;
}
```

***Listado 3-10:** Declarando reglas con múltiples selectores*

También podemos referenciar solo elementos que se encuentran dentro de un elemento en particular listando los selectores separados por un espacio. Estos tipos de selectores se llaman *selectores de descendencia* porque afectan a elementos dentro de otros elementos, sin importar el lugar que ocupan en la jerarquía.

```
main p {
  font-size: 20px;
}
```

***Listado 3-11:** Combinando selectores*

La regla en el Listado 3-11 afecta solo a los elementos **<p>** que se encuentran dentro de un elemento **<main>**, ya sea como contenido directo o insertados en otros elementos. Por ejemplo, el siguiente documento incluye un sección principal con una cabecera y una sección adicional. Ambos elementos incluyen elementos **<p>** para representar su contenido. Si aplicamos la regla del Listado 3-11 a este documento, el texto dentro de cada elemento **<p>** se mostrará con un tamaño de 20 píxeles porque todos son descendientes del elemento **<main>**.

```
<!DOCTYPE html>
<html lang="es">
```

```
<head>
  <title>Este texto es el título del documento</title>
  <meta charset="utf-8">
  <link rel="stylesheet" href="misestilos.css">
</head>

<body>
  <main>
    <header>
      <h1>Título</h1>
      <p>Esta es la introducción</p>
    </header>
    <section>
      <p>Frase 1</p>
      <p>Frase 2</p>
      <p>Frase 3</p>
      <p>Frase 4</p>
    </section>
  </main>
</body>
</html>
```

Listado 3-12: Probando selectores

La regla del Listado 3-11 solo afecta a elementos <p> que se encuentran dentro del elemento <main>. Si, por ejemplo, agregamos un elemento <footer> al final del documento del Listado 3-12, los elementos <p> dentro de este pie de página no se verán modificados. La Figura 3-2 muestra lo que vemos cuando abrimos este documento en el navegador.

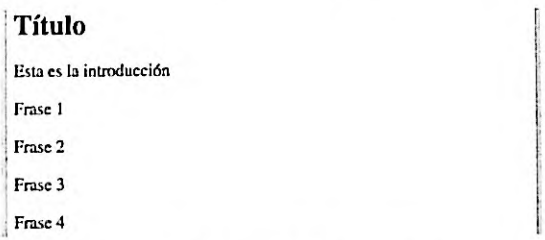

Figura 3-2: Selectores de descendencia

CSS incluye sintaxis adicionales para realizar una selección más precisa. Por ejemplo, podemos usar el carácter > para referenciar un elemento que es hijo directo de otro elemento.

```
section > p {
  font-size: 20px;
}
```

Listado 3-13: Aplicando el selector >

El carácter > indica que el elemento afectado es el elemento de la derecha cuando tiene como padre al elemento de la izquierda. La regla del Listado 3-13 modifica los elementos <p>

que son hijos de un elemento <section>. Cuando aplicamos esta regla al documento del Listado 3-12, el elemento <p> dentro del elemento <header> no se ve afectado.

Figura 3-3: Selector de hijo

Con el carácter + se crea otro selector que facilita CSS. Este selector referencia un elemento que está precedido por otro elemento. Ambos deben compartir el mismo elemento padre.

```
h1 + p {
  font-size: 20px;
}
```

Listado 3-14: Aplicando el selector +

La regla del Listado 3-14 afecta a todos los elementos <p> que se ubican inmediatamente después de un elemento <h1>. Si aplicamos esta regla a nuestro documento, solo se modificará el elemento <p> dentro de la cabecera porque es el único que está precedido por un elemento <h1>.

Figura 3-4: Selector del elemento adyacente

El selector anterior afecta solo al elemento <p> que se ubica inmediatamente después de un elemento <h1>. Si se coloca otro elemento entre ambos elementos, el elemento <p> no se modifica. CSS incluye el carácter ~ para crear un selector que afecta a todos los elementos que se ubican a continuación de otro elemento. Este selector es similar al anterior, pero el elemento afectado no tiene que encontrarse inmediatamente después del primer elemento. Esta regla afecta a todos los elementos encontrados, no solo al primero.

```
p ~ p {
  font-size: 20px;
}
```

Listado 3-15: Aplicando el selector ~

La regla del Listado 3-15 afecta a todos los elementos <p> que preceden a otro elemento <p>. En nuestro ejemplo, se modificarán todos los elementos <p> dentro del elemento <section>, excepto el primero, porque no existe un elemento <p> que preceda al primer elemento <p>.

Figura 3-5: Selector general de hermano

Hágalo usted mismo: cree un nuevo archivo HTML con el código del Listado 3-12. Cree un archivo CSS llamado misestilos.css y escriba dentro la regla que quiere probar. Inserte otros elementos entre los elementos <p> para ver cómo trabajan estas reglas.

Atributo Id

Las reglas anteriores afectan a los elementos del tipo indicado por el selector. Para seleccionar un elemento HTML sin considerar su tipo, podemos usar el atributo `id`. Este atributo es un nombre, un identificador exclusivo del elemento y, por lo tanto, lo podemos usar para encontrar un elemento en particular dentro del documento. Para referenciar un elemento usando su atributo `id`, el selector debe incluir el valor del atributo precedido por el carácter numeral (#).

```
#mitexto {
  font-size: 20px;
}
```

Listado 3-16: Referenciando por medio del valor del atributo `id`

La regla del Listado 3-16 solo se aplica al elemento identificado por el atributo `id` y el valor "mitexto", como el elemento <p> incluido en el siguiente documento.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Este texto es el título del documento</title>
  <meta charset="utf-8">
  <link rel="stylesheet" href="misestilos.css">
</head>
<body>
  <main>
    <section>
      <p>Frase 1</p>
      <p id="mitexto">Frase 2</p>
      <p>Frase 3</p>
```

```
      <p>Frase 4</p>
    </section>
  </main>
</body>
</html>
```

Listado 3-17: Identificando un elemento `<p>` *por medio de su atributo* `id`

La ventaja de este procedimiento es que cada vez que creamos una referencia usando el identificador **mitexto** en nuestro archivo CSS, solo se modifica el elemento con esa identificación, pero el resto de los elementos no se ven afectados. Esta es una forma muy específica de referenciar a un elemento y se usa comúnmente con elementos estructurales, como **<section>** o ** **. Debido a su especificidad, el atributo **id** también se usa frecuentemente para referenciar elementos desde JavaScript, tal como veremos en los próximos capítulos.

Atributo Class

En lugar de usar el atributo **id** para asignar estilos, en la mayoría de las ocasiones es mejor hacerlo con el atributo **class**. Este atributo es más flexible y se puede asignar a varios elementos dentro del mismo documento.

```
.mitexto {
  font-size: 20px;
}
```

Listado 3-18: Referenciando por medio del valor del atributo `class`

Para referenciar un elemento usando su atributo **class**, el selector debe incluir el valor del atributo precedido por un punto. Por ejemplo, la regla del Listado 3-18 afecta a todos los elementos que contienen un atributo **class** con el valor "mitexto", como en el siguiente ejemplo.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Este texto es el título del documento</title>
  <meta charset="utf-8">
  <link rel="stylesheet" href="misestilos.css">
</head>

<body>
  <main>
    <section>
      <p class="mitexto">Frase 1</p>
      <p class="mitexto">Frase 2</p>
      <p>Frase 3</p>
      <p>Frase 4</p>
    </section>
  </main>
</body>
</html>
```

Listado 3-19: Asignando estilos con el atributo `class`

Los elementos <p> de las dos primeras líneas dentro de la sección principal del documento del Listado 3-19 incluyen el atributo `class` con el valor "mitexto". Debido a que se puede aplicar la misma regla a diferentes elementos del documento, estos dos primeros elementos son afectados por la regla del Listado 3-18. Por otro lado, los dos últimos elementos <p> no incluyen el atributo `class` y, por lo tanto, se mostrarán con los estilos por defecto.

Las reglas asignadas a través del atributo `class` se denominan *clases*. A un mismo elemento se le pueden asignar varias clases. Todo lo que tenemos que hacer es declarar los nombres de las clases separados por un espacio (por ejemplo, `class="texto1 texto2"`). Las clases también se pueden declarar como exclusivas para un tipo específico de elementos declarando el nombre del elemento antes del punto.

```
p.mitexto {
  font-size: 20px;
}
```

***Listado 3-20:** Declarando una clase solo para elementos* <p>

En el Listado 3-20, hemos creado una regla que referencia la clase llamada `mitexto`, pero solo para los elementos <p>. Otros elementos que contengan el mismo valor en su atributo `class` no se verán afectados por esta regla.

Otros atributos

Aunque estos métodos de referencia cubren una amplia variedad de situaciones, a veces son insuficientes para encontrar el elemento exacto que queremos modificar. Para esas situaciones en las que los atributos `id` y `class` no son suficientes, CSS nos permite referenciar un elemento por medio de cualquier otro atributo que necesitemos. La sintaxis para definir esta clase de selectores incluye el nombre del elemento seguido del nombre del atributo en corchetes.

```
p[name] {
  font-size: 20px;
}
```

***Listado 3-21:** Referenciando solo elementos* <p> *que tienen un atributo* `name`

La regla del Listado 3-21 modifica solo los elementos <p> que tienen un atributo llamado `name`. Para emular lo que hemos hecho con los atributos `id` y `class`, también podemos incluir el valor del atributo.

```
p[name="mitexto"] {
  font-size: 20px;
}
```

***Listado 3-22:** Referenciando elementos* <p> *que tienen un atributo* `name` *con el valor* `mitexto`

CSS nos permite combinar el carácter = con otros caracteres para realizar una selección más específica. Los siguientes caracteres son los más utilizados.

```
p[name-="mi"] {
  font-size: 20px;
}
p[name^="mi"] {
  font-size: 20px;
}
p[name$="mi"] {
  font-size: 20px;
}
p[name*="mi"] {
  font-size: 20px;
}
```

Listado 3-23: Aplicando selectores de atributo

Las reglas del Listado 3-23 se han construido con selectores que incluyen los mismos atributos y valores, pero referencian diferentes elementos, como se describe a continuación.

- El selector con los caracteres -= referencia cualquier elemento <p> con un atributo `name` cuyo valor incluye la palabra "`mi`" (por ejemplo, "`mi texto`", "`mi coche`").
- El selector con los caracteres ^= referencia cualquier elemento <p> con el atributo `name` cuyo valor comienza en "`mi`" (por ejemplo, "`mitexto`", "`micoche`").
- El selector con los caracteres $= referencia cualquier elemento <p> con un atributo `name` cuyo valor termina en "`mi`" (por ejemplo, "`textomi`", "`cochemi`").
- El selector con los caracteres *= referencia cualquier elemento <p> con un atributo `name` cuyo valor contiene la cadena de caracteres "`mi`" (en este caso, la cadena de caracteres podría también encontrarse en el medio del texto, como en "`textomicoche`").

En estos ejemplos, usamos el elemento <p>, el atributo `name` y un texto arbitrario como "`mi`", pero se puede aplicar la misma técnica a cualquier atributo y valor que necesitemos.

Seudoclases

Las seudoclases son herramientas especiales de CSS que nos permiten referenciar elementos HTML por medio de sus características, como sus posiciones en el código o sus condiciones actuales. Las siguientes son las que más se utilizan.

:nth-child(valor**)**—Esta seudoclase selecciona un elemento de una lista de elementos hermanos que se encuentra en la posición especificada por el valor entre paréntesis.

:first-child—Esta seudoclase selecciona el primer elemento de una lista de elementos hermanos.

:last-child—Esta seudoclase selecciona el último elemento de una lista de elementos hermanos.

:only-child—Esta seudoclase selecciona un elemento cuando es el único hijo de otro elemento.

:first-of-type—Esta seudoclase selecciona el primer elemento de una lista de elementos del mismo tipo.

:not(selector**)**—Esta seudoclase selecciona los elementos que no coinciden con el selector entre paréntesis.

Con estos selectores, podemos realizar una selección más dinámica. En los siguientes ejemplos, vamos a aplicar algunos utilizando un documento sencillo.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Este texto es el título del documento</title>
  <meta charset="utf-8">
  <link rel="stylesheet" href="misestilos.css">
</head>
<body>
  <main>
    <section>
      <p class="mitexto1">Frase 1</p>
      <p class="mitexto2">Frase 2</p>
      <p class="mitexto3">Frase 3</p>
      <p class="mitexto4">Frase 4</p>
    </section>
  </main>
</body>
</html>
```

Listado 3-24: Creando un documento para probar las seudoclases

La sección principal del documento del Listado 3-24 incluye cuatro elementos **<p>** que, considerando la estructura HTML, son hermanos y, por lo tanto, hijos del mismo elemento **<section>**. Usando seudoclases, podemos aprovechar esta organización y referenciar elementos sin importar cuánto conocemos acerca de sus atributos o valores. Por ejemplo, con la seudoclase **:nth-child()** podemos modificar todos los segundos elementos **<p>** que se encuentran en el documento.

```
p:nth-child(2) {
  font-size: 20px;
}
```

Listado 3-25: Implementando la seudoclase **:nth-child()**

Lo básico: las seudoclases se pueden agregar a cualquiera de los selectores que hemos mencionado con anterioridad. En la regla del Listado 3-25 hemos referenciado los elementos **<p>** usando el nombre **p**, pero esta regla también se podría haber escrito como **.miclase:nth-child(2)**, por ejemplo, para referenciar todo elemento que es hijo de otro elemento y que incluye el atributo **class** con el valor "miclase".

Lo que indica la seudoclase **:nth-child()** es algo como "el hijo en la posición...", por lo que el número entre paréntesis corresponde al número de posición del elemento hijo, o índice. Usando este tipo de referencias, podemos, por supuesto, seleccionar cualquier elemento hijo que queramos simplemente cambiando el número de índice. Por ejemplo, la siguiente regla afectará solo al último elemento <p> de nuestro documento.

```
p:nth-child(4) {
  font-size: 20px;
}
```

Listado 3-26: Cambiando el índice para afectar a un elemento diferente

Es posible asignar estilos a cada elemento creando una regla similar para cada uno de ellos.

```
p:nth-child(1) {
  background-color: #999999;
}
p:nth-child(2) {
  background-color: #CCCCCC;
}
p:nth-child(3) {
  background-color: #999999;
}
p:nth-child(4) {
  background-color: #CCCCCC;
}
```

Listado 3-27: Generando una lista con la seudoclase **:nth-child()**

En el Listado 3-27, hemos usado la seudoclase **:nth-child()** y la propiedad **background-color** para generar una lista de opciones que son claramente diferenciadas por el color de fondo. El valor **#999999** define un gris oscuro y el valor **#CCCCCC** define un gris claro. Por lo tanto, el primer elemento tendrá un fondo oscuro, el segundo un fondo claro, y así sucesivamente.

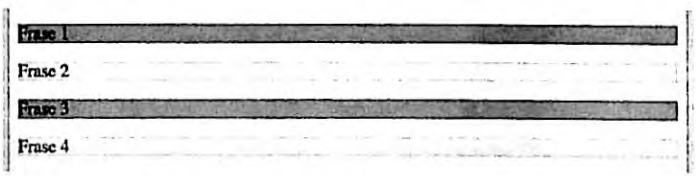

Figura 3-6: La seudoclase **:nth-child()**

Se podrían agregar más opciones a la lista incorporando nuevos elementos <p> en el código HTML y nuevas reglas con la seudoclase **:nth-child()** en la hoja de estilo. Sin embargo, esta técnica nos obligaría a extender el código innecesariamente y sería imposible aplicarla en sitios web con contenido dinámico. Una alternativa es la que ofrecen las palabras clave **odd** y **even** disponibles para esta seudoclase.

```
p:nth-child(odd) {
  background-color: #999999;
}
p:nth-child(even) {
  background-color: #CCCCCC;
}
```

Listado 3-28: Implementando las palabras clave odd *y* even

La palabra clave **odd** para la seudoclase **:nth-child()** afecta a los elementos **<p>** que son hijos de otro elemento y tienen un índice impar, y la palabra clave **even** afecta a aquellos que tienen un índice par. Usando estas palabras clave, solo necesitamos dos reglas para configurar la lista completa, sin importar lo larga que sea. Incluso cuantas más opciones o filas se incorporen más adelante al documento, menos necesario será agregar otras reglas al archivo CSS. Los estilos se asignarán automáticamente a cada elemento de acuerdo con su posición.

Además de esta seudoclase existen otras que están relacionadas y también nos pueden ayudar a encontrar elementos específicos dentro de la jerarquía, como **:first-child**, **:last-child** y **:only-child**. La seudoclase **:first-child** referencia solo el primer elemento hijo, **:last-child** referencia solo el último elemento hijo, y **:only-child** afecta a un elemento cuando es el único elemento hijo. Estas seudoclases no requieren palabras clave o ningún parámetro adicional y se implementan según muestra el siguiente ejemplo.

```
p:last-child {
  font-size: 20px;
}
```

Listado 3-29: Usando `:last-child` *para modificar el último elemento* <p> *de la lista*

Otra seudoclase importante es **:not()**. Con esta seudoclase podemos seleccionar elementos que no coinciden con un selector. Por ejemplo, la siguiente regla asigna un margen de 0 píxeles a todos los elementos con nombres diferentes de **p**.

```
:not(p) {
  margin: 0px;
}
```

Listado 3-30: Aplicando estilos a todos los elementos excepto <p>

En lugar del nombre del elemento podemos usar cualquier otro selector que necesitemos. En el siguiente listado, se verán afectados todos los elementos dentro del elemento **<section>** excepto aquellos que contengan el atributo **class** con el valor "mitexto2".

```
section :not(.mitexto2) {
  margin: 0px;
}
```

Listado 3-31: Definiendo una excepción por medio del atributo `class`

Cuando aplicamos esta última regla al código HTML del Listado 3-24, el navegador asigna los estilos por defecto al elemento <p> con la clase `mitexto2` y asigna un margen de 0 píxeles al resto (el elemento <p> con la clase `mitexto2` es el que no se verá afectado por la regla). El resultado se ilustra en la Figura 3-7. Los elementos <p> primero, tercero y cuarto se presentan sin margen, pero el segundo elemento <p> se muestra con los márgenes superiores e inferiores por defecto.

Figura 3-7: La seudoclase `:not()`

3.3 Propiedades

Las propiedades son la pieza central de CSS. Todos los estilos que podemos aplicar a un elemento se definen por medio de propiedades. Ya hemos introducido algunas en los ejemplos anteriores, pero hay cientos de propiedades disponibles. Para simplificar su estudio, se pueden clasificar en dos tipos: propiedades de formato y propiedades de diseño. Las propiedades de formato se encargan de dar forma a los elementos y su contenido, mientras que las de diseño están enfocadas a determinar el tamaño y la posición de los elementos en la pantalla. Así mismo, las propiedades de formato se pueden clasificar según el tipo de modificación que producen. Por ejemplo, algunas propiedades cambian el tipo de letra que se usa para mostrar el texto, otras generan un borde alrededor del elemento, asignan un color de fondo, etc. En este capítulo vamos a introducir las propiedades de formato siguiendo esta clasificación.

Texto

Desde CSS se pueden controlar varios aspectos del texto, como el tipo de letra que se usa para mostrar en pantalla, el espacio entre líneas, la alineación, etc. Las siguientes son las propiedades disponibles para definir el tipo de letra, tamaño, y estilo de un texto.

font-family—Esta propiedad declara el tipo de letra que se usa para mostrar el texto. Se pueden declarar múltiples valores separados por coma para ofrecer al navegador varias alternativas en caso de que algunos tipos de letra no se encuentren disponibles en el ordenador del usuario. Algunos de los valores estándar son `Georgia`, `"Times New Roman"`, `Arial`, `Helvetica`, `"Arial Black"`, `Gadget`, `Tahoma`, `Geneva`, `Helvetica`, `Verdana`, `Geneva`, `Impact`, y `sans-serif` (los nombres compuestos por más de una palabra se deben declarar entre comillas dobles).

font-size—Esta propiedad determina el tamaño de la letra. El valor puede ser declarado en píxeles (`px`), porcentaje (`%`), o usando cualquiera de las unidades disponibles en CSS como `em`, `rem`, `pt`, etc. El valor por defecto es normalmente `16px`.

font-weight—Esta propiedad determina si el texto se mostrará en negrita o no. Los valores disponibles son `normal` y `bold`, pero también podemos asignar los valores `100`, `200`, `300`, `400`, `500`, `600`, `700`, `800`, y `900` para determinar el grosor de la letra (solo disponibles para algunos tipos de letra).

font-style—Esta propiedad determina el estilo de la letra. Los valores disponibles son `normal`, `italic`, y `oblique`.

font—Esta propiedad nos permite declarar múltiples valores al mismo tiempo. Los valores deben declararse separados por un espacio y en un orden preciso. El estilo y el grosor se deben declarar antes que el tamaño, y el tipo de letra al final (por ejemplo, `font: bold 24px Arial,sans-serif`).

Estas propiedades se deben aplicar a cada elemento (o elemento padre) cuyo texto queremos modificar. Por ejemplo, el siguiente documento incluye una cabecera con un título y una sección con un párrafo. Como queremos asignar diferentes tipos de letra al título y al texto, tenemos que incluir el atributo `id` en cada elemento para poder identificarlos desde las reglas CSS.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Este texto es el título del documento</title>
  <meta charset="utf-8">
  <link rel="stylesheet" href="misestilos.css">
</head>
<body>
  <header>
    <span id="titulo">Hojas de Estilo en Cascada</span>
  </header>
  <section>
    <p id="descripcion">Hojas de estilo en cascada (o CSS, siglas en
inglés de Cascading Style Sheets) es un lenguaje de diseño gráfico para
definir y crear la presentación de un documento estructurado escrito en
HTML.</p>
  </section>
  <footer>
    <a href="http://www.jdgauchat.com">www.jdgauchat.com</a>
  </footer>
</body>
</html>
```

Listado 3-32: Probando propiedades de formato

Las reglas deben incluir todas las propiedades requeridas para asignar los estilos que deseamos. Por ejemplo, si queremos asignar un tipo de letra y un nuevo tamaño al texto, tenemos que incluir las propiedades `font-family` y `font-size`.

```
#titulo {
  font-family: Verdana, sans-serif;
  font-size: 26px;
}
```

Listado 3-33: Cambiando el tipo de letra y el tamaño del título

Aunque podemos declarar varias propiedades en la misma regla para modificar diferentes aspectos del elemento, CSS ofrece una propiedad abreviada llamada `font` que nos permite

definir todas las características del texto en una sola línea de código. Los valores asignados a esta propiedad se deben declarar separados por un espacio. Por ejemplo, la siguiente regla asigna el estilo **bold** (negrita), un tamaño de 26 píxeles, y el tipo de letra **Verdana** al elemento **titulo**.

```
#titulo {
  font: bold 26px Verdana, sans-serif;
}
```

Listado 3-34: Cambiando el tipo de letra con la propiedad `font`

Cuando la regla del Listado 3-34 se aplica al documento del Listado 3-32, el título se muestra con los valores definidos por la propiedad **font** y el párrafo se presenta con los estilos por defecto.

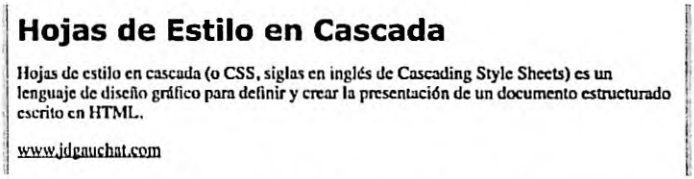

Figura 3-8: La propiedad `font`

Hágalo usted mismo: cree un nuevo archivo HTML con el código del Listado 3-32. Cree o actualice el archivo misestilos.css con la regla del Listado 3-34. Abra el documento en su navegador. Debería ver algo parecido a lo que se muestra en la Figura 3-8.

Desde CSS podemos cambiar no solo el tipo de letra, sino también otros aspectos del texto, como el alineamiento, la sangría, el espacio entre líneas, etc. Las siguientes son algunas de las propiedades disponibles para este propósito.

text-align—Esta propiedad alinea el texto dentro de un elemento. Los valores disponibles son **left**, **right**, **center**, y **justify**.

text-align-last—Esta propiedad alinea la última línea de un párrafo. Los valores disponibles son **left**, **right**, **center**, y **justify**.

text-indent—Esta propiedad define el tamaño de la sangría de un párrafo (el espacio vacío al comienzo de la línea). El valor se puede declarar en pixeles (**px**), porcentaje (**%**), o usando cualquiera de las unidades disponibles en CSS, como **em**, **rem**, **pt**, etc.

letter-spacing—Esta propiedad define el espacio entre letras. El valor se debe declarar en píxeles (**px**), porcentaje (**%**) o usando cualquiera de las unidades disponibles en CSS, como **em**, **rem**, **pt**, etc.

word-spacing—Esta propiedad define el ancho del espacio entre palabras. El valor puede ser declarado en píxeles (**px**), porcentaje (**%**) o usando cualquiera de las unidades disponibles en CSS, como **em**, **rem**, **pt**, etc.

line-height—Esta propiedad define el espacio entre líneas. El valor se puede declarar en píxeles (`px`), porcentaje (`%`) o usando cualquiera de las unidades disponibles en CSS, como `em`, `rem`, `pt`, etc.

vertical-align—Esta propiedad alínea elementos verticalmente. Se usa frecuentemente para alinear texto con imágenes (la propiedad se aplica a la imagen). Los valores disponibles son `baseline`, `sub`, `super`, `text-top`, `text-bottom`, `middle`, `top`, y `bottom`.

Por defecto, los navegadores alinean el texto a la izquierda, pero podemos cambiar la alineación con la propiedad `text-align`. La siguiente regla centra el párrafo de nuestro documento.

```
#titulo {
  font: bold 26px Verdana, sans-serif;
}
#descripcion {
  text-align: center;
}
```

Listado 3-35: Alineando el texto con la propiedad `text-align`

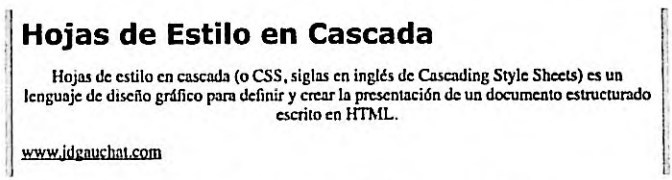

Figura 3-9: La propiedad `text-align`

El resto de las propiedades listadas arriba son simples de aplicar. Por ejemplo, podemos definir el tamaño del espacio entre palabras con la propiedad `word-spacing`.

```
#titulo {
  font: bold 26px Verdana, sans-serif;
}
#descripcion {
  text-align: center;
  word-spacing: 20px;
}
```

Listado 3-36: Definiendo el espacio entre palabras con la propiedad `word-spacing`

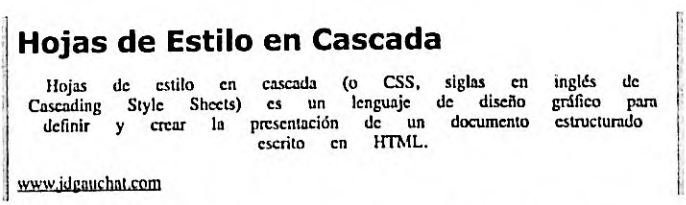

Figura 3-10: La propiedad `word-spacing`

CSS también ofrece la siguiente propiedad para decorar el texto con una línea.

text-decoration—Esta propiedad resalta el texto con una línea. Los valores disponibles son `underline`, `overline`, `line-through`, y `none`.

La propiedad `text-decoration` es particularmente útil con enlaces. Por defecto, los navegadores muestran los enlaces subrayados. Si queremos eliminar la línea, podemos declarar esta propiedad con el valor `none`. El siguiente ejemplo agrega una regla a nuestra hoja de estilo para modificar los elementos `<a>` del documento.

```
#titulo {
  font: bold 26px Verdana, sans-serif;
}
#descripcion {
  text-align: center;
}
a {
  text-decoration: none;
}
```

Listado 3-37: Eliminando las líneas en los enlaces de nuestro documento

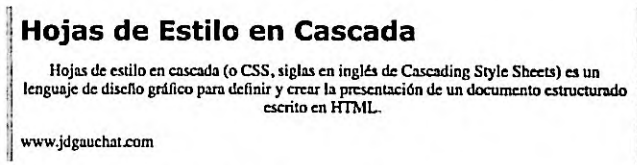

Figura 3-11: La propiedad `text-decoration`

Hasta el momento, hemos utilizado el tipo de letra `Verdana` (y la alternativa `sans-serif` en caso de que `Verdana` no esté disponible). Este tipo de letra es parte de un grupo conocido como *fuentes seguras* porque se encuentran disponibles en la mayoría de los ordenadores y, por lo tanto, podemos usarla con seguridad. El problema con las fuentes en la Web es que los navegadores no las descargan desde el servidor del sitio web, las cargan desde el ordenador del usuario, y los usuarios tienen diferentes tipos de letra instaladas en sus sistemas. Si usamos una fuente que el usuario no posee, el diseño de nuestro sitio web se verá diferente. Usando fuentes seguras nos aseguramos de que nuestro sitio web se verá de igual manera en cualquier navegador u ordenador, pero el problema es que los tipos de letras incluidos en este grupo son pocos. Para ofrecer una solución y permitir plena creatividad al diseñador, CSS incluye la regla `@font-face`. La regla `@font-face` es una regla reservada que permite a los diseñadores incluir un archivo con la fuente de letra a usar para mostrar el texto de una página web. Si la usamos, podemos incluir cualquier fuente que deseemos en nuestro sitio web con solo facilitar el archivo que la contiene.

La regla `@font-face` necesita al menos dos propiedades para declarar la fuente y cargar el archivo. La propiedad `font-family` especifica el nombre que queremos usar para referenciar este tipo de letra y la propiedad `src` indica la URL del archivo con las especificaciones de la fuente (la sintaxis de la propiedad `src` requiere el uso de la función `url()` para indicar la URL del archivo). En el siguiente ejemplo, el nombre `MiNuevaLetra` se asigna a nuestra fuente y el archivo `font.ttf` se indica como el archivo a cargar.

```
#titulo {
  font: bold 26px MiNuevaLetra, Verdana, sans-serif;
}
@font-face {
  font-family: "MiNuevaLetra";
  src: url("font.ttf");
}
```

Listado 3-38: Cargando un tipo de letra personalizado para el título

Una vez que se carga la fuente, podemos usarla en cualquier elemento del documento por medio de su nombre (`MiNuevaLetra`). En la propiedad `font` de la regla `#titulo` del Listado 3-38, especificamos que el título se mostrará con la nueva fuente o con las fuentes alternativas `Verdana` y `sans-serif` si nuestra fuente no se ha cargado.

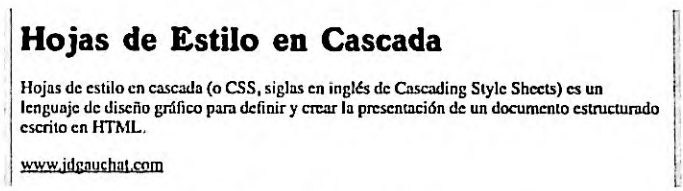

Figura 3-12: Título con un tipo de letra personalizado

Hágalo usted mismo: descargue el archivo font.ttf desde nuestro sitio web. Copie el archivo dentro del directorio de su proyecto. Actualice su hoja de estilo con el código del Listado 3-38 y abra el documento del Listado 3-32 en su navegador. Puede encontrar más fuentes como la que se muestra en la Figura 3-12 en www.moorstation.org/typoasis/designers/steffmann/.

Lo básico: una función es un trozo de código que realiza una tarea específica y devuelve el resultado. La función `url()`, por ejemplo, tiene la tarea de cargar el archivo indicado entre paréntesis y devolver su contenido. CSS incluye varias funciones para generar los valores de sus propiedades. Introduciremos algunas de estas funciones a continuación y aprenderemos más sobre funciones en el Capítulo 6.

Colores

Existen dos formas de declarar un color en CSS: podemos usar una combinación de tres colores básicos (rojo, verde y azul), o definir el matiz, la saturación y la luminosidad. El color final se crea considerando los niveles que asignamos a cada componente. Dependiendo del tipo de sistema que utilizamos para definir el color, tendremos que declarar los niveles usando números hexadecimales (desde 00 a FF), números decimales (desde 0 a 255) o porcentajes. Por ejemplo, si decidimos usar una combinación de niveles de rojo, verde y azul, podemos declarar los niveles con números hexadecimales. En este caso, los valores del color se declaran en secuencia y precedidos por el carácter numeral, como en #996633 (99 es el nivel de rojo, 66

es el nivel de verde y 33 es el nivel de azul). Para definir colores con otros tipos de valores, CSS ofrece las siguientes funciones.

rgb(rojo, verde, azul)—Esta función define un color por medio de los valores especificados por los atributos (desde 0 a 255). El primer valor representa el nivel de rojo, el segundo valor representa el nivel de verde y el último valor el nivel de azul (por ejemplo, `rgb(153, 102, 51)`).

rgba(rojo, verde, azul, alfa)—Esta función es similar a la función `rgb()`, pero incluye un componente adicional para definir la opacidad (alfa). El valor se puede declarar entre 0 y 1, con 0 como totalmente transparente y 1 como totalmente opaco.

hsl(matiz, saturación, luminosidad)—Esta función define un color desde los valores especificados por los atributos. Los valores se declaran en números decimales y porcentajes.

hsla(matiz, saturación, luminosidad, alfa)—Esta función es similar a la función `hsl()`, pero incluye un componente adicional para definir la opacidad (alfa). El valor se puede declarar entre 0 y 1, con 0 como totalmente transparente y 1 como totalmente opaco.

Como veremos más adelante, son varias las propiedades que requieren valores que definen colores, pero la siguiente es la que se utiliza con más frecuencia:

color—Esta propiedad declara el color del contenido del elemento.

La siguiente regla asigna un gris claro al título de nuestro documento usando números hexadecimales.

```
#titulo {
  font: bold 26px Verdana, sans-serif;
  color: #CCCCCC;
}
```

***Listado 3-39:** Asignando un color al título*

Cuando los niveles de rojo, verde y azul son iguales, como en este caso, el color final se encuentra dentro de una escala de grises, desde negro (#000000) a blanco (#FFFFFF).

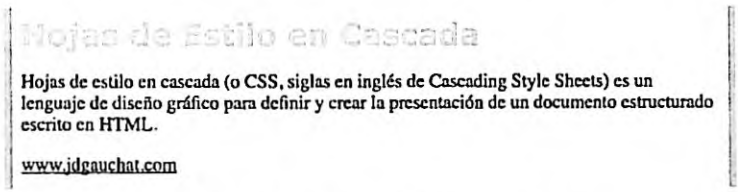

***Figura 3-13:** Título con un color personalizado*

Declarar un color con una función es bastante parecido: solo tenemos que reemplazar el valor hexadecimal por la función que queremos utilizar. Por ejemplo, podemos definir el mismo color de grises con la función `rgb()` y valores decimales.

```
#titulo {
  font: bold 26px Verdana, sans-serif;
  color: rgb(204, 204, 204);
}
```

Listado 3-40: Asignando un color con la función `rgb()`

La función `hsl()` es simplemente otra función disponible para generar un color, pero es más intuitiva que `rgb()`. A algunos diseñadores les resulta más fácil crear grupos de colores usando `hsl()`. Los valores requeridos por esta función definen el matiz, la saturación y la luminosidad. El matiz es un color extraído de una rueda imaginaria, expresado en grados desde 0 a 360: alrededor de 0 y 360 se encuentran los rojos, cerca de 120 los verdes, y cerca de 240 los azules. La saturación se representa en porcentaje, desde 0 % (escala de grises) a 100 % (todo color o totalmente saturado), y la luminosidad es también un valor en porcentaje, desde 0 % (completamente negro) a 100 % (completamente blanco); un valor de 50 % representa una luminosidad promedio. Por ejemplo, en la siguiente regla, la saturación se define como 0 % para crear un color dentro de la escala de grises y la luminosidad se especifica en 80 % para obtener un gris claro.

```
#titulo {
  font: bold 26px Verdana, sans-serif;
  color: hsl(0, 0%, 80%);
}
```

Listado 3-41: Asignando un color con la función `hsl()`

IMPORTANTE: CSS define una propiedad llamada `opacity` para declarar la opacidad de un elemento. Esta propiedad presenta el problema de que el valor de opacidad para un elemento lo heredan sus elementos hijos. Ese problema se resuelve con las funciones `rgba()` y `hsla()`, con las que podemos asignar un valor de opacidad al fondo de un elemento, como veremos a continuación, sin que su contenido se vea afectado.

Lo básico: no es práctico encontrar los colores adecuados para nuestro sitio web combinando números y valores. Para facilitar esta tarea, los ordenadores personales incluyen varias herramientas visuales que nos permiten seleccionar un color y obtener su valor. Además, la mayoría de los editores de fotografía y programas gráficos disponibles en el mercado incluyen una herramienta que muestra una paleta desde donde podemos seleccionar un color y obtener el correspondiente valor hexadecimal o decimal. Para encontrar los colores adecuados, puede usar estas herramientas o cualquiera de las disponibles en la Web, como **www.colorhexa.com** o **htmlcolorcodes.com**.

Tamaño

Por defecto, el tamaño de la mayoría de los elementos se determina según el espacio disponible en el contenedor. El ancho de un elemento se define como 100 %, lo cual significa que será tan ancho como su contenedor, y tendrá una altura determinada por su contenido.

Esta es la razón por la que el elemento <p> del documento del Listado 3-32 se extendía a los lados de la ventana del navegador y no era más alto de lo necesario para contener las líneas del párrafo. CSS define las siguientes propiedades para declarar un tamaño personalizado:

width—Esta propiedad declara el ancho de un elemento. El valor se puede especificar en píxeles, porcentaje, o con la palabra clave `auto` (por defecto). Cuando el valor se especifica en porcentaje, el ancho se calcula según el navegador a partir del ancho del contenedor, y cuando se declara con el valor `auto`, el elemento se expande hasta ocupar todo el espacio horizontal disponible dentro del contenedor.

height—Esta propiedad declara la altura de un elemento. El valor se puede especificar en píxeles, porcentaje, o con la palabra clave `auto` (por defecto). Cuando el valor se especifica en porcentaje, el navegador calcula la altura a partir de la altura del contenedor, y cuando se declara con el valor `auto`, el elemento adopta la altura de su contenedor.

Los navegadores generan una caja alrededor de cada elemento que determina el área que ocupa en la pantalla. Cuando declaramos un tamaño personalizado, la caja se modifica y el contenido del elemento se adapta para encajar dentro de la nueva área, tal como muestra la Figura 3-14.

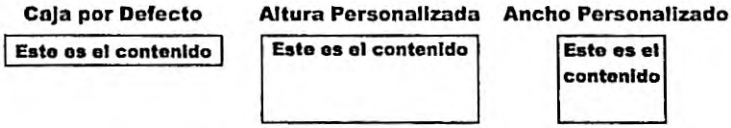

Figura 3-14: Caja personalizada

Por ejemplo, si declaramos un ancho de 200 píxeles para el elemento <p> de nuestro documento, las líneas del texto serán menos largas, pero se agregarán nuevas líneas para mostrar todo el párrafo.

```
#titulo {
  font: bold 26px Verdana, sans-serif;
}
#descripcion {
  width: 200px;
}
```

Listado 3-42: Asignando un ancho personalizado

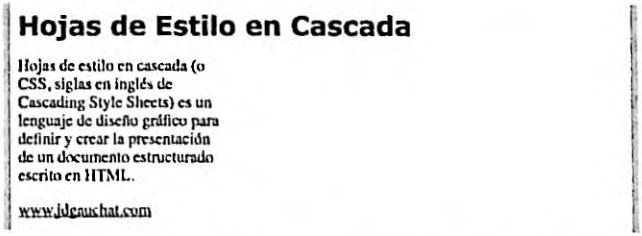

Figura 3-15: Contenido principal con un tamaño personalizado

La regla del Listado 3-42 declara el ancho del elemento <p>, pero la altura queda aún determinada por su contenido, lo que significa que la caja generada por este elemento será más alta para contener el párrafo completo, según ilustra la Figura 3-15. Si también queremos restringir la altura del elemento, podemos usar la propiedad **height**. La siguiente regla reduce la altura del elemento <p> de nuestro documento a 100 píxeles.

```
#titulo {
  font: bold 26px Verdana, sans-serif;
}
#descripcion {
  width: 200px;
  height: 100px;
}
```

Listado 3-43: Asignando una altura personalizada

El problema con elementos que tienen un tamaño definido es que a veces el contenido no se puede mostrar en su totalidad. Por defecto, los navegadores muestran el resto del contenido fuera del área de la caja. En consecuencia, parte del contenido de una caja con tamaño personalizado se puede posicionar sobre el contenido del elemento que se encuentra debajo, tal como se ilustra en la Figura 3-16.

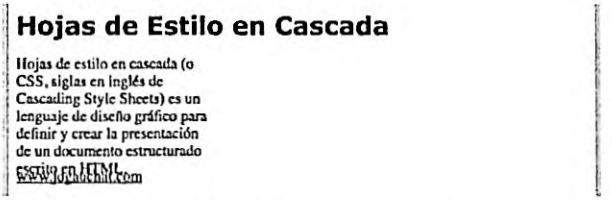

Figura 3-16: Contenido desbordado

En nuestro ejemplo, el texto que se encuentra fuera de la caja determinada por el elemento <p> ocupa el espacio correspondiente al elemento **<footer>** y, por lo tanto, el contenido de ambos elementos se encuentra superpuesto. CSS incluye las siguientes propiedades para resolver este problema:

overflow—Esta propiedad especifica cómo se mostrará el contenido que desborda el elemento. Los valores disponibles son **visible** (por defecto), **hidden** (esconde el contenido que no entra dentro de la caja), **scroll** (muestra barras laterales para desplazar el contenido), **auto** (deja que el navegador decida qué hacer con el contenido).

overflow-x—Esta propiedad especifica cómo se mostrará el contenido que desborda el elemento horizontalmente. Acepta los mismos valores que la propiedad **overflow**.

overflow-y—Esta propiedad especifica cómo se mostrará el contenido que desborda el elemento verticalmente. Acepta los mismos valores que la propiedad **overflow**.

overflow-wrap—Esta propiedad indica si un palabra debería ser dividida en un punto arbitrario cuando no hay suficiente espacio para mostrarla en la línea. Los valores disponibles son **normal** (la línea será dividida naturalmente) y **break-word** (las palabras se dividirán en puntos arbitrarios para acomodar la línea de texto en el espacio disponible).

Con estas propiedades podemos determinar cómo se mostrará el contenido cuando no hay suficiente espacio disponible. Por ejemplo, podemos ocultar el texto que desborda el elemento asignando el valor **hidden** a la propiedad **overflow**.

```
#titulo {
  font: bold 26px Verdana, sans-serif;
}
#descripcion {
  width: 200px;
  height: 100px;
  overflow: hidden;
}
```

Listado 3-44: Ocultando el contenido que desborda el elemento

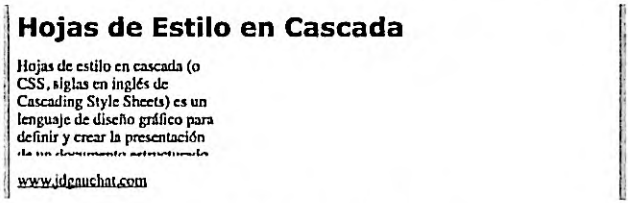

Figura 3-17: Contenido ocultado

Si queremos que el usuario pueda ver el texto que se ha ocultado, podemos asignar el valor **scroll** y forzar al navegador a mostrar barras laterales para desplazar el contenido.

```
#titulo {
  font: bold 26px Verdana, sans-serif;
}
#descripcion {
  width: 200px;
  height: 100px;
  overflow: scroll;
}
```

Listado 3-45: Incorporando barras de desplazamiento

Figura 3-18: Barras de desplazamiento

El tamaño del elemento no queda solo determinado por el ancho y la altura de su caja, sino también por el relleno y los márgenes. CSS nos permite designar espacio alrededor de la caja

para separar el elemento de otros elementos a su alrededor (margen), además de incluir espacio entre los límites de la caja y su contenido (relleno). La Figura 3-19 ilustra cómo se aplican estos espacios a un elemento.

Figura 3-19: Márgenes, rellenos y bordes

CSS incluye las siguientes propiedades para definir márgenes y rellenos para un elemento.

margin—Esta propiedad declara el margen de un elemento. El margen es el espacio que hay alrededor de la caja. Puede recibir cuatro valores que representan el margen superior, derecho, inferior, e izquierdo, en ese orden y separados por un espacio (por ejemplo, `margin: 10px 30px 10px 30px;`). Sin embargo, si solo se declaran uno, dos o tres valores, los otros toman los mismos valores (por ejemplo, `margin: 10px 30px` asigna 10 píxeles al margen superior e inferior y 30 píxeles al margen izquierdo y derecho). Los valores se pueden declarar independientemente usando las propiedades asociadas `margin-top`, `margin-right`, `margin-bottom` y `margin-left` (por ejemplo, `margin-left: 10px;`). La propiedad también acepta el valor `auto` para obligar al navegador a calcular el margen (usado para centrar un elemento dentro de su contenedor).

padding—Esta propiedad declara el relleno de un elemento. El relleno es el espacio entre el contenido del elemento y los límites de su caja. Los valores se declaran de la misma forma que lo hacemos para la propiedad `margin`, aunque también se pueden declarar de forma independiente con las propiedades `padding-top`, `padding-right`, `padding-bottom` y `padding-left` (por ejemplo, `padding-top: 10px;`).

La siguiente regla agrega márgenes y relleno a la cabecera de nuestro documento. Debido a que asignamos solo un valor, el mismo valor se usa para definir todos los márgenes y rellenos del elemento (superior, derecho, inferior e izquierdo, en ese orden).

```
header {
  margin: 30px;
  padding: 15px;
}
#titulo {
  font: bold 26px Verdana, sans-serif;
}
```

Listado 3-46: Agregando márgenes y relleno

El tamaño de un elemento y sus márgenes se agregan para calcular el área que ocupa. Lo mismo pasa con el relleno y el borde (estudiaremos bordes más adelante). El tamaño final de un elemento se calcula con la fórmula: tamaño + márgenes + relleno + bordes. Por ejemplo, si tenemos un elemento con un ancho de 200 píxeles y un margen de 10 píxeles a cada lado, el

ancho del área ocupada por el elemento será de 220 píxeles. El total de 20 píxeles de margen se agrega a los 200 píxeles del elemento y el valor final se representa en la pantalla.

Figura 3-20: Cabecera con márgenes y relleno personalizados

Hágalo usted mismo: reemplace las reglas en su archivo CSS con las reglas del Listado 3-46 y abra el documento en su navegador. Debería ver algo similar a lo que se representa en la Figura 3-20. Cambie el valor de la propiedad `margin` para ver cómo afecta a los márgenes.

IMPORTANTE: como veremos más adelante en este capítulo, los elementos se clasifican en dos tipos principales: Block (bloque) e Inline (en línea). Los elementos Block pueden tener un tamaño personalizado, pero los elementos Inline solo pueden ocupar el espacio determinado por sus contenidos. El elemento `<span>` que usamos para definir el título de la cabecera se declara por defecto como elemento Inline y, por lo tanto, no puede tener un tamaño y unos márgenes personalizados. Esta es la razón por la que en nuestro ejemplo asignamos márgenes y relleno al elemento `<header>` en lugar de al elemento `<span>` (los elementos estructurales se definen todos como elementos Block).

Fondo

Los elementos pueden incluir un fondo que se muestra detrás del contenido del elemento y a través del área ocupada por el contenido y el relleno. Debido a que el fondo puede estar compuesto por colores e imágenes, CSS define varias propiedades para generarlo.

background-color—Esta propiedad asigna un fondo de color a un elemento.

background-image—Esta propiedad asigna una o varias imágenes al fondo de un elemento. La URL del archivo se declara con la función `url()` (por ejemplo, `url("ladrillos.jpg")`). Si se requiere más de una imagen, los valores se deben separar por una coma.

background-position—Esta propiedad declara la posición de comienzo de una imagen de fondo. Los valores se pueden especificar en porcentaje, píxeles o usando una combinación de las palabras clave `center`, `left`, `right`, `top`, y `bottom`.

background-size—Esta propiedad declara el tamaño de la imagen de fondo. Los valores se pueden especificar en porcentaje, píxeles, o usando las palabras clave `cover` y `contain`. La palabra clave `cover` expande la imagen hasta que su ancho o su altura cubren el área del elemento, mientras que `contain` estira la imagen para ocupar toda el área del elemento.

background-repeat—Esta propiedad determina cómo se distribuye la imagen de fondo usando cuatro palabras clave: `repeat`, `repeat-x`, `repeat-y` y `no-repeat`. La palabra clave `repeat` repite la imagen en el eje vertical y horizontal, mientras que `repeat-x` y

`repeat-y` lo hacen solo en el eje horizontal o vertical, respectivamente. Finalmente, `no-repeat` muestra la imagen de fondo una sola vez.

background-origin—Esta propiedad determina si la imagen de fondo se posicionará considerando el borde, el relleno o el contenido del área del elemento. Los valores disponibles son `border-box`, `padding-box`, y `content-box`.

background-clip—Esta propiedad declara el área a cubrir por el fondo usando los valores `border-box`, `padding-box`, y `content-box`. El primer valor corta la imagen al borde de la caja del elemento, el segundo corta la imagen en el relleno de la caja y el tercero corta la imagen alrededor del contenido de la caja.

background-attachment—Esta propiedad determina si la imagen es estática o se desplaza con el resto de los elementos usando dos valores: `scroll` (por defecto) y `fixed`. El valor `scroll` hace que la imagen se desplace con la página, y el valor `fixed` fija la imagen de fondo en su lugar original.

background—Esta propiedad nos permite declarar todos los atributos del fondo al mismo tiempo.

Los fondos más comunes se crean con colores. La siguiente regla implementa la propiedad `background-color` para agregar un fondo gris a la cabecera de nuestro documento.

```
header {
  margin: 30px;
  padding: 15px;
  text-align: center;
  background-color: #CCCCCC;
}
#titulo {
  font: bold 26px Verdana, sans-serif;
}
```

Listado 3-47: Agregando un color de fondo

Debido a que definimos márgenes de 30 píxeles a los lados, la cabecera queda centrada en la ventana del navegador, pero el texto dentro del elemento `<header>` aún se encuentra alineado a la izquierda (la alineación por defecto). En el ejemplo del Listado 3-47, además de cambiar el fondo, también incluimos la propiedad `text-align` para centrar el título. El resultado se muestra en la Figura 3-21.

Figura 3-21: Fondo de color

Además de colores, también podemos usar imágenes de fondo. En este caso, tenemos que declarar el fondo con la propiedad `background-image` y declarar la URL de la imagen con la función `url()`, como lo hacemos en el siguiente ejemplo.

```
header {
  margin: 30px;
  padding: 15px;
  text-align: center;
  background-image: url("ladrillosclaros.jpg");
}
#titulo {
  font: bold 26px Verdana, sans-serif;
}
```

Listado 3-48: Agregando una imagen de fondo

El problema con las imágenes es que no siempre son del mismo tamaño que la caja creada por el elemento. Por esta razón, los navegadores repiten la imagen una y otra vez hasta cubrir toda el área. El resultado se muestra en la Figura 3-22.

Figura 3-22: Imagen de fondo

Hágalo usted mismo: reemplace las reglas en su archivo CSS por las reglas del Listado 3-48. Descargue la imagen ladrillosclaros.jpg desde nuestro sitio web. Copie la imagen en el mismo directorio donde se encuentra su documento. Abra el documento en su navegador. Debería ver algo parecido a lo que se muestra en la Figura 3-22.

Si queremos modificar el comportamiento por defecto, podemos usar el resto de las propiedades de fondo disponibles. Por ejemplo, si asignamos el valor `repeat-y` a la propiedad `background-repeat`, el navegador solo repetirá la imagen en el eje vertical.

```
header {
  margin: 30px;
  padding: 15px;
  text-align: center;
  background-image: url("ladrillosclaros.jpg");
  background-repeat: repeat-y;
}
#titulo {
  font: bold 26px Verdana, sans-serif;
}
```

Listado 3-49: Configurando el fondo

Figura 3-23: Fondo de imagen

Cuando nuestro diseño requiere varios valores para configurar el fondo, podemos declararlos todos juntos con la propiedad **background**. Esta propiedad nos permite declarar diferentes aspectos del fondo en una sola línea de código.

```
header {
  margin: 30px;
  padding: 15px;
  text-align: center;
  background: #CCCCCC url("ladrillosclaros.jpg") repeat-y;
}
#titulo {
  font: bold 26px Verdana, sans-serif;
}
```


Listado 3-50: Configurando el fondo con la propiedad `background`

La regla del Listado 3-50 especifica los mismos valores que usamos anteriormente, pero ahora combina una imagen de fondo con un color. El resultado se muestra en la Figura 3-24.

Figura 3-24: Imagen de fondo combinada con un color de fondo

Bordes

Los elementos pueden incluir un borde en los límites de la caja del elemento. Por defecto, los navegadores no muestran ningún borde, pero podemos usar las siguientes propiedades para definirlo.

border-width—Esta propiedad define el ancho del borde. Acepta hasta cuatro valores separados por un espacio para especificar el ancho de cada lado del borde (superior, derecho, inferior, e izquierdo, es ese orden). También podemos declarar el ancho para cada lado de forma independiente con las propiedades **border-top-width**, **border-bottom-width**, **border-left-width**, y **border-right-width**.

border-style—Esta propiedad define el estilo del borde. Acepta hasta cuatro valores separados por un espacio para especificar los estilos de cada lado del borde (superior, derecho, inferior, e izquierdo, en ese orden). Los valores disponibles son **none**, **hidden**, **dotted**, **dashed**, **solid**, **double**, **groove**, **ridge**, **inset**, y **outset**. El valor por defecto es **none**, lo que significa que el borde no se mostrará a menos que asignemos un valor diferente a esta propiedad. También podemos declarar los estilos de forma independiente con las propiedades **border-top-style**, **border-bottom-style**, **border-left-style**, y **border-right-style**.

border-color—Esta propiedad define el color del borde. Acepta hasta cuatro valores separados por un espacio para especificar el color de cada lado del borde (superior, derecho, inferior, e izquierdo, en ese orden). También podemos declarar los colores de forma independiente con las propiedades **border-top-color**, **border-bottom-color**, **border-left-color**, y **border-right-color**.

border—Esta propiedad nos permite declarar todos los atributos del borde al mismo tiempo. También podemos usar las propiedades **border-top**, **border-bottom**, **border-left**, y **border-right** para definir los valores de cada borde de forma independiente.

Para asignar un borde a un elemento, todo lo que tenemos que hacer es definir el estilo con la propiedad **border-style**. Una vez que se define el estilo, el navegador usa los valores por defecto para generar el borde. Si no queremos dejar que el navegador determine estos valores, podemos usar el resto de las propiedades para configurar todos los atributos del borde. El siguiente ejemplo asigna un borde sólido de 2 píxeles de ancho a la cabecera de nuestro documento.

```
header {
  margin: 30px;
  padding: 15px;
  text-align: center;
  border-style: solid;
  border-width: 2px;
}
#titulo {
  font: bold 26px Verdana, sans-serif;
}
```

Listado 3-51: Agregando un borde a un elemento

Figura 3-25: Borde sólido

Como hemos hecho con la propiedad **background**, declaramos todos los valores juntos usando la propiedad **border**. El siguiente ejemplo crea un borde discontinuo de 5 píxeles en color gris.

```
header {
  margin: 30px;
  padding: 15px;
  text-align: center;
  border: 5px dashed #CCCCCC;
}
#titulo {
  font: bold 26px Verdana, sans-serif;
}
```

Listado 3-52: Definiendo un borde con la propiedad **border**

Figura 3-26: Borde discontinuo

El borde agregado con estas propiedades se dibuja alrededor de la caja del elemento, lo que significa que va a describir un rectángulo con esquinas rectas. Si nuestro diseño requiere esquinas redondeadas, podemos agregar la siguiente propiedad.

border-radius—Esta propiedad define el radio del círculo virtual que el navegador utilizará para dibujar las esquinas redondeadas. Acepta hasta cuatro valores para definir los radios de cada esquina (superior izquierdo, superior derecho, inferior derecho e inferior izquierdo, en ese orden). También podemos usar las propiedades `border-top-left-radius`, `border-top-right-radius`, `border-bottom-right-radius`, y `border-bottom-left-radius` para definir el radio de cada esquina de forma independiente.

El siguiente ejemplo genera esquinas redondeadas para nuestra cabecera con un radio de 20 píxeles.

```
header {
  margin: 30px;
  padding: 15px;
  text-align: center;
  border: 2px solid;
  border-radius: 20px;
}
#titulo {
  font: bold 26px Verdana, sans-serif;
}
```

Listado 3-53: Generando esquinas redondeadas

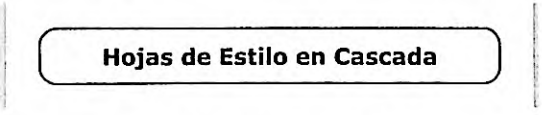

Figura 3-27: Esquinas redondeadas

Si todas las esquinas son iguales, podemos declarar solo un valor para esta propiedad, tal como lo hemos hecho en el ejemplo anterior. Sin embargo, al igual que con las propiedades `margin` y `padding`, si queremos que las esquinas sean diferentes, tenemos que especificar valores diferentes para cada una de ellas.

```
header {
  margin: 30px;
  padding: 15px;
  text-align: center;
  border: 2px solid;
  border-radius: 20px 10px 30px 50px;
}
#titulo {
  font: bold 26px Verdana, sans-serif;
}
```

Listado 3-54: Declarando diferentes valores para cada esquina

En el Listado 3-54, los valores asignados a la propiedad **border-radius** representan cuatro ubicaciones diferentes. Los valores se declaran siempre en la dirección de las agujas del reloj, comenzando por la esquina superior izquierda. El orden es: esquina superior izquierda, esquina superior derecha, esquina inferior derecha y esquina inferior izquierda.

Figura 3-28: Diferentes esquinas

Lo básico: al igual que las propiedades **margin** y **padding**, la propiedad **border-radius** también puede aceptar solo dos valores. El primer valor se asigna a las esquinas primera y tercera (superior izquierdo e inferior derecho) y a las esquinas segunda y cuarta (superior derecho e inferior izquierdo).

También podemos cambiar la forma de las esquinas agregando valores separados por una barra oblicua. Los valores de la izquierda representan el radio horizontal y los valores de la derecha representan el radio vertical. La combinación de estos valores genera una elipse.

```
header {
  margin: 30px;
  padding: 15px;
  text-align: center;
  border: 2px solid;
  border-radius: 20px / 10px;
}
#titulo {
  font: bold 26px Verdana, sans-serif;
}
```

Listado 3-55: Generando esquinas elípticas

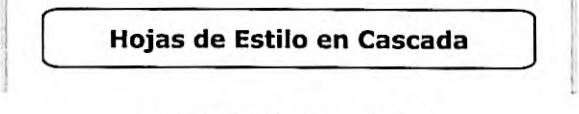

Figura 3-29: Esquinas elípticas

Hágalo usted mismo: copie los estilos que desea probar dentro de su archivo CSS y abra el documento en su navegador. Modifique los valores de cada propiedad pare entender cómo trabajan.

Los bordes que acabamos de crear se dibujan en los límites de la caja del elemento, pero también podemos demarcar el elemento con un segundo borde que se dibuja alejado de esto límites. El propósito de estos tipos de bordes es resaltar el elemento. Algunos navegadores lo usan para demarcar texto, pero la mayoría dibuja un segundo borde fuera de los límites de la caja. CSS ofrece las siguientes propiedades para crear este segundo borde.

outline-width—Esta propiedad define el ancho del borde. Acepta valores en cualquiera de las unidades disponibles en CSS (`px`, `%`, `em`, etc.) y también las palabras clave `thin`, `medium`, y `thick`.

outline-style—Esta propiedad define el estilo del borde. Los valores disponibles son `none`, `auto`, `dotted`, `dashed`, `solid`, `double`, `groove`, `ridge`, `inset`, y `outset`.

outline-color—Esta propiedad define el color del borde.

outline-offset—Esta propiedad define el desplazamiento (a qué distancia de los límites de la caja se dibujará el segundo borde). Acepta valores en cualquiera de las unidades disponibles en CSS (`px`, `%`, `em`, etc.).

outline—Esta propiedad nos permite especificar el ancho, estilo y color del borde al mismo tiempo (el desplazamiento aún se debe definir con la propiedad `outline-offset`).

Por defecto, el desplazamiento se declara con el valor 0, por lo que el segundo borde se dibujará a continuación del borde de la caja. Si queremos separar los dos bordes, tenemos que definir la propiedad `outline-offset`.

```
header {
  margin: 30px;
  padding: 15px;
  text-align: center;
  border: 1px solid #999999;
  outline: 2px dashed #000000;
  outline-offset: 15px;
}
#titulo {
  font: bold 26px Verdana, sans-serif;
}
```

Listado 3-56: Agregando un segundo borde a la cabecera

En el Listado 3-56, agregamos un segundo borde de 2 píxeles con un desplazamiento de 15 píxeles a los estilos originales asignados a la caja de la cabecera de nuestro documento. El resultado se muestra en la Figura 3-30.

Figura 3-30: Segundo borde

Los efectos logrados por las propiedades `border` y `outline` se limitan a simples líneas y unas pocas opciones de configuración, pero CSS nos permite definir bordes personalizados usando imágenes para superar estas limitaciones. Las siguientes son las propiedades que se incluyen con este propósito.

border-image-source—Esta propiedad determina la imagen que se usará para crear el borde. La URL del archivo se declara con la función `url()` (por ejemplo, `url("ladrillos.jpg")`).

border-image-width—Esta propiedad define el ancho del borde. Acepta valores en cualquiera de las unidades disponibles en CSS (`px`, `%`, `em`, etc.).

border-image-repeat—Esta propiedad define cómo se usa la imagen para generar el borde. Los valores disponibles son `repeat`, `round`, `stretch`, y `space`.

border-image-slice—Esta propiedad define cómo se va a cortar la imagen para representar las esquinas del borde. Debemos asignar cuatro valores para especificar los cuatro trozos de la imagen que se utilizarán (si solo se declara un valor, se usa para todos los lados). El valor se puede especificar como un entero o en porcentaje.

border-image-outset—Esta propiedad define el desplazamiento del borde (la distancia a la que se encuentra de la caja del elemento). Acepta valores en cualquiera de las unidades disponibles en CSS (`px`, `%`, `em`, etc.).

border-image—Esta propiedad nos permite especificar todos los atributos del borde al mismo tiempo.

Estas propiedades usan una imagen como patrón. De acuerdo a los valores facilitados, la imagen se corta como un pastel para obtener las piezas necesarias y luego estas piezas se acomodan alrededor del elemento para construir el borde.

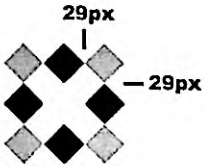

Figura 3-31: Patrón para construir el borde

Para lograr este objetivo, necesitamos declarar tres atributos: la ubicación del archivo de la imagen, el tamaño de las piezas que queremos extraer del patrón y las palabras clave que determinan cómo se van a distribuir estas piezas alrededor del elemento.

```
header {
  margin: 30px;
  padding: 15px;
  text-align: center;
  border: 29px solid;
  border-image-source: url("diamantes.png");
  border-image-slice: 29;
  border-image-repeat: stretch;
}
#titulo {
  font: bold 26px Verdana, sans-serif;
}
```

Listado 3-57: Creando un borde personalizado para la caja de la cabecera

En el Listado 3-57, creamos un borde de 29 píxeles para la caja de la cabecera y luego cargamos la imagen diamantes.png para construir el borde. El valor 29 asignado a la propiedad `border-image-slice` declara el tamaño de las piezas, y el valor `stretch`, asignado a la propiedad `border-image-repeat`, es uno de los métodos disponibles para distribuir estas piezas alrededor de la caja. Hay tres valores disponibles para este último atributo. El valor `repeat` repite las piezas tomadas de la imagen cuantas veces sea necesario para cubrir el lado del elemento. En este caso, el tamaño de la pieza se preserva y la imagen solo se corta si no hay espacio suficiente para ubicarla. El valor `round` calcula la longitud del lado de la caja y luego estira las piezas para asegurarse de que no se corta ninguna. Finalmente, el valor `stretch` (usado en el Listado 3-57) estira una pieza hasta cubrir todo el lado.

Figura 3-32: Borde de tipo `stretch`

Como siempre, podemos declarar todos los valores al mismo tiempo usando una sola propiedad.

```
header {
  margin: 30px;
  padding: 15px;
  text-align: center;
  border: 29px solid;
  border-image: url("diamantes.png") 29 round;
}
#titulo {
  font: bold 26px Verdana, sans-serif;
}
```

Listado 3-58: Definiendo el borde con la propiedad `border-image`

Figura 3-33: Borde de tipo `round`

Sombras

Otro efecto interesante que podemos aplicar a un elemento son las sombras. CSS incluye las siguientes propiedades para generar sombras para la caja de un elemento y también para formas irregulares como texto.

box-shadow—Esta propiedad genera una sombra desde la caja del elemento. Acepta hasta seis valores. Podemos declarar el desplazamiento horizontal y vertical de la sombra, el radio de difuminado, el valor de propagación, el color de la sombra, y también podemos incluir el valor `inset` para indicar que la sombra deberá proyectarse dentro de la caja.

text-shadow—Esta propiedad genera una sombra desde un texto. Acepta hasta cuatro valores. Podemos declarar el desplazamiento horizontal y vertical, el radio de difuminado y el color de la sombra.

La propiedad **box-shadow** necesita al menos tres valores para poder determinar el color y el desplazamiento de la sombra. El desplazamiento puede ser positivo o negativo. Los valores indican la distancia horizontal y vertical desde la sombra al elemento: los valores negativos posicionan la sombra a la izquierda y encima del elemento, mientras que los positivos crean una sombra a la derecha y debajo del elemento. El valor 0 ubica a la sombra detrás del elemento y ofrece la posibilidad de generar un efecto de difuminado a su alrededor. El siguiente ejemplo agrega una sombra básica a la cabecera de nuestro documento.

```
header {
  margin: 30px;
  padding: 15px;
  text-align: center;
  border: 1px solid;
  box-shadow: rgb(150,150,150) 5px 5px;
}
#titulo {
  font: bold 26px Verdana, sans-serif;
}
```

Listado 3-59: Agregando una sombra a la cabecera

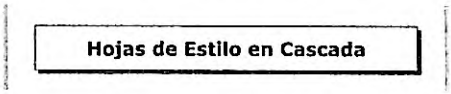

Figura 3-34: Sombra básica

La sombra que hemos obtenido hasta el momento es sólida, sin gradientes ni transparencia, pero aún no se parece a una sombre real. Para mejorar su aspecto podemos agregar una distancia de difuminado.

```
header {
  margin: 30px;
  padding: 15px;
  text-align: center;
  border: 1px solid;
  box-shadow: rgb(150,150,150) 5px 5px 20px;
}
#titulo {
  font: bold 26px Verdana, sans-serif;
}
```

Listado 3-60: Agregando el valor de difuminado con la propiedad `box-shadow`

Figura 3-35: Sombra

Al agregar otro valor en píxeles al final de la propiedad, podemos propagar la sombra. Este efecto cambia levemente la apariencia de la sombra y expande el área que ocupa.

```
header {
  margin: 30px;
  padding: 15px;
  text-align: center;
  border: 1px solid;
  box-shadow: rgb(150,150,150) 10px 10px 20px 10px;
}
#titulo {
  font: bold 26px Verdana, sans-serif;
}
```

Listado 3-61: Expandiendo la sombra

Figura 3-36: Sombra más amplia

El último valor disponible para la propiedad **box-shadow** no es un número, sino la palabra clave **inset**. Este valor transforma la sombra externa en una sombra interna, lo cual otorga un efecto de profundidad al elemento.

```
header {
  margin: 30px;
  padding: 15px;
  text-align: center;
  border: 1px solid;
  box-shadow: rgb(150,150,150) 5px 5px 10px inset;
}
#titulo {
  font: bold 26px Verdana, sans-serif;
}
```

Listado 3-62: Creando una sombra interna

Figura 3-37: Sombra interna

IMPORTANTE: las sombras no expanden el elemento ni tampoco incrementan su tamaño, por lo que deberá asegurarse de que haya suficiente espacio alrededor del elemento para que se vea la sombra.

La propiedad **box-shadow** se ha diseñado específicamente para cajas de elementos. Si intentamos aplicar este efecto a un elemento ****, por ejemplo, la sombra se asignará a la caja alrededor del elemento, no a su contenido. CSS define una propiedad aparte para generar la sombra de un texto llamada **text-shadow**.

```
header {
  margin: 30px;
  padding: 15px;
  text-align: center;
  border: 1px solid;
}
#titulo {
  font: bold 26px Verdana, sans-serif;
  text-shadow: rgb(150, 150, 150) 3px 3px 5px;
}
```

***Listado 3-63:** Agregando una sombra al título*

Los valores de la propiedad **text-shadow** son similares a los asignados a la propiedad **box-shadow**. Podemos especificar el color de la sombra, la distancia horizontal y vertical desde la sombra al elemento, y el radio de difuminado. En el Listado 3-63, se genera una sombra para el título de la cabecera con una distancia de 3 píxeles y un radio de difuminado de 5. El resultado se muestra en la Figura 3-38.

***Figura 3-38:** Sombra de texto*

Gradientes

Un gradiente se forma mediante una serie de colores que varían continuamente con una transición suave de un color a otro. Los gradientes se crean como imágenes y se agregan al fondo del elemento con las propiedades **background-image** o **background**. Para crear la imagen con el gradiente, CSS ofrece las siguientes funciones:

linear-gradient(posición, ángulo, colores**)**—Esta función crea un gradiente lineal. El atributo **posición** determina el lado o la esquina desde la cual comienza el gradiente y se declara con los valores **top**, **bottom**, **left** y **right**; el atributo **ángulo** define la dirección del gradiente y se puede declarar en las unidades **deg** (grados), **grad** (gradianes), **rad** (radianes), o **turn** (espiras), y el atributo **colores** es la lista de colores que participan en el gradiente separados por coma. Los valores para el atributo **colores** pueden incluir un segundo valor en porcentaje separado por un espacio para indicar la posición donde finaliza el color.

radial-gradient(posición, forma, colores, extensión)—Esta función crea un gradiente radial. El atributo **posición** indica el origen del gradiente y se puede declarar en píxeles, en porcentaje, o por medio de la combinación de los valores `center`, `top`, `bottom`, `left` y `right`, el atributo **forma** determina la forma del gradiente y se declara con los valores `circle` y `ellipse`, el atributo **colores** es la lista de los colores que participan en el gradiente separados por coma, y el atributo **extensión** determina la forma que el gradiente va a adquirir con los valores `closest-side`, `closest-corner`, `farthest-side`, y `farthest-corner`. Los valores para el atributo **colores** pueden incluir un segundo valor en porcentaje separado por un espacio para indicar la posición donde finaliza el color.

Los gradientes se declaran como imágenes de fondo, por lo que podemos aplicarlos a un elemento por medio de las propiedades `background-image` o `background`, como lo hacemos en el siguiente ejemplo.

```
header {
  margin: 30px;
  padding: 15px;
  text-align: center;
  border: 1px solid;
  background: -webkit-linear-gradient(top, #FFFFFF, #666666);
}
#titulo {
  font: bold 26px Verdana, sans-serif;
}
```

Listado 3-64: Agregando un gradiente lineal a la cabecera

Figura 3-39: Gradiente lineal

IMPORTANTE: algunas propiedades y funciones CSS todavía se consideran experimentales. Por esta razón, se deben declarar con un prefijo que representa el motor web utilizado. Por ejemplo, si queremos que la función `linear-gradient()` funcione en Google Chrome, tenemos que declararla como `-webkit-linear-gradient()`. Si quiere usar esta función en su sitio web, deberá repetir la propiedad `background` para cada navegador existente con su correspondiente prefijo. Los prefijos requeridos para los navegadores más populares son `-moz-` para Mozilla Firefox, `-webkit-` para Safari y Google Chrome, `-o-` para Opera, y `-ms-` para Internet Explorer.

En el ejemplo anterior, usamos el valor `top` para determinar la posición inicial del gradiente, pero también podemos combinar dos valores para comenzar el gradiente desde una esquina del elemento, como en el siguiente ejemplo.

```
header {
  margin: 30px;
  padding: 15px;
  text-align: center;
  border: 1px solid;
  background: -webkit-linear-gradient(top right, #FFFFFF, #666666);
}
#titulo {
  font: bold 26px Verdana, sans-serif;
}
```

Listado 3-65: Estableciendo la posición inicial

Figura 3-40: Diferente comienzo para un gradiente lineal

Cuando trabajamos con gradientes lineales, también podemos configurar la dirección con un ángulo en grados.

```
header {
  margin: 30px;
  padding: 15px;
  text-align: center;
  border: 1px solid;
  background: -webkit-linear-gradient(30deg, #FFFFFF, #666666);
}
#titulo {
  font: bold 26px Verdana, sans-serif;
}
```

Listado 3-66: Creando un gradiente con una dirección de 30 grados

Figura 3-41: Gradiente lineal con la dirección establecida en grados

Los gradientes anteriores se han creado con solo dos colores, pero podemos agregar más valores para generar un gradiente multicolor.

```
header {
  margin: 30px;
  padding: 15px;
  text-align: center;
  border: 1px solid;
  background: -webkit-linear-gradient(top, #000000, #FFFFFF, #999999);
```

```
}
#titulo {
  font: bold 26px Verdana, sans-serif;
}
```

Listado 3-67: Creando un gradiente multicolor

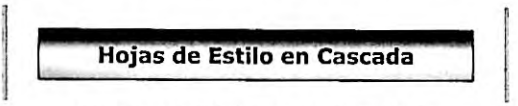

Figura 3-42: Gradiente lineal multicolor

Si usamos el valor **transparent** en lugar de un color, podemos hacer que el gradiente sea traslúcido y de esta manera se mezcle con el fondo (este efecto también se puede lograr con la función **rgba()** estudiada anteriormente). En el siguiente ejemplo, asignamos una imagen de fondo al elemento **<body>** para cambiar el fondo de la página y poder comprobar que logramos un gradiente translúcido.

```
body {
  background: url("ladrillosclaros.jpg");
}
header {
  margin: 30px;
  padding: 15px;
  text-align: center;
  border: 1px solid;
  background: -webkit-linear-gradient(top, transparent, #666666);
}
#titulo {
  font: bold 26px Verdana, sans-serif;
}
```

Listado 3-68: Creando un gradiente translúcido

Figura 3-43: Gradiente translúcido

Los parámetros que definen los colores también pueden determinar el punto de comienzo y final de cada color incluyendo un valor adicional en porcentaje.

```
header {
  margin: 30px;
  padding: 15px;
  text-align: center;
  border: 1px solid;
  background: -webkit-linear-gradient(top, #FFFFFF 50%, #666666 90%);
}
```

```
#titulo {
  font: bold 26px Verdana, sans-serif;
}
```

Listado 3-69: Configurando los puntos de comienzo y final de cada color

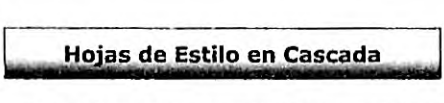

Figura 3-44: Gradiente lineal con valores de comienzo y final

Además de gradientes lineales, también podemos crear gradientes radiales. La sintaxis para gradientes radiales no difiere mucho de los gradientes lineales que acabamos de estudiar. La única diferencia es que tenemos que usar la función **radial-gradient()** en lugar de la función **linear-gradient()** e incluir un parámetro que determina la forma del gradiente con los valores **circle** o **ellipse**.

```
header {
  margin: 30px;
  padding: 15px;
  text-align: center;
  border: 1px solid;
  background: -webkit-radial-gradient(center, ellipse, #FFFFFF,
#000000);
}
#titulo {
  font: bold 26px Verdana, sans-serif;
}
```

Listado 3-70: Creando un gradiente radial

Figura 3-45: Gradiente radial

Excepto por la forma (círculo o elipse), el resto de esta función trabaja del mismo modo que **linear-gradient()**. La posición del gradiente se puede personalizar y podemos usar varios colores con un segundo valor para determinar los límites de cada uno de ellos.

```
header {
  margin: 30px;
  padding: 15px;
  text-align: center;
  border: 1px solid;
  background: -webkit-radial-gradient(30px 50px, ellipse, #FFFFFF 50%,
#666666 70%, #999999 90%);
}
```

```
#titulo {
  font: bold 26px Verdana, sans-serif;
}
```

Listado 3-71: Creando un gradiente radial multicolor

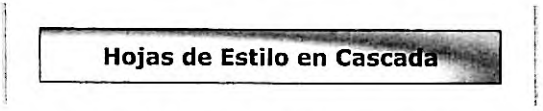

Figura 3-46: Gradiente radial con puntos de inicio y final

Filtros

Los filtros agregan efectos a un elemento y su contenido. CSS incluye la propiedad `filter` para asignar un filtro a un elemento y las siguiente funciones para crearlo.

blur(valor)—Esta función produce un efecto de difuminado. Acepta valores en píxeles desde `1px` a `10px`.

grayscale(value)—Esta función convierte los colores de la imagen en una escala de grises. Acepta números decimales entre `0.1` y `1`.

drop-shadow(x, y, tamaño, color)—Esta función genera una sombra. Los atributos **x** e **y** determinan la distancia entre la sombra y la imagen, el atributo **tamaño** especifica el tamaño de la sombra, y el atributo **color** declara su color.

sepia(valor)—Esta función le otorga un tono sepia (ocre) a los colores de la imagen. Acepta números decimales entre `0.1` y `1`.

brightness(valor)—Esta función cambia el brillo de la imagen. Acepta números decimales entre `0.1` y `10`.

contrast(valor)—Esta función cambia el contraste de la imagen. Acepta números decimales entre `0.1` y `10`.

hue-rotate(valor)—Esta función aplica una rotación a los matices de la imagen. Acepta un valor en grados desde `1deg` a `360deg`.

invert(valor)—Esta función invierte los colores de la imagen y produce un negativo. Acepta números decimales entre `0.1` y `1`.

saturate(valor)—Esta función satura los colores de la imagen. Acepta números decimales entre `0.1` y `10`.

opacity(valor)—Esta función cambia la opacidad de la imagen. Acepta números decimales entre `0` y `1` (`0` es totalmente transparente y `1` totalmente opaco).

Estos filtros no solo se pueden aplicar a imágenes, sino también a otros elementos en el documento. El siguiente ejemplo aplica un efecto de difuminado a la cabecera de nuestro documento.

```
header {
  margin: 30px;
  padding: 15px;
  text-align: center;
  border: 1px solid;
  filter: blur(5px);
}
#titulo {
  font: bold 26px Verdana, sans-serif;
}
```

Listado 3-72: Aplicando un filtro a la cabecera

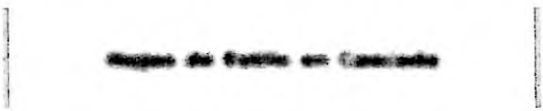

Figura 3-47: Una cabecera borrosa

Hágalo usted mismo: reemplace las reglas en su archivo CSS por las reglas del Listado 3-72 y abra el documento en su navegador. Reemplace la función `blur()` con cualquiera de las restantes funciones disponibles para ver cómo trabajan.

Transformaciones

Una vez que se crean los elementos HTML, estos permanecen inmóviles, pero podemos modificar su posición, tamaño y apariencia por medio de la propiedad `transform`. Esta propiedad realiza cuatro transformaciones básicas a un elemento: escalado, rotación, inclinación y traslación. Las siguientes son las funciones definidas para este propósito.

scale(x, y)—Esta función modifica la escala del elemento. Existen otras dos funciones relacionadas llamadas `scaleX()` y `scaleY()` pare especificar los valores horizontales y verticales independientemente.

rotate(ángulo)—Esta función rota el elemento. El atributo representa los grados de rotación y se puede declarar en `deg` (grados), `grad` (gradianes), `rad` (radianes) o `turn` (espiras).

skew(ángulo)—Esta función inclina el elemento. El atributo representa los grados de desplazamiento. La función puede incluir dos valores para representar el ángulo horizontal y vertical. Los valores se pueden declarar en `deg` (grados), `grad` (gradianes), `rad` (radianes) o `turn` (espiras).

translate(x, y)—Esta función desplaza al elemento a la posición determinada por los atributos **x** e **y**.

La función `scale()` recibe dos parámetros, el valor **x** para la escala horizontal y el valor **y** para la escala vertical. Si solo se declara un valor, ese mismo valor se usa para ambos parámetros.

```
header {
  margin: 30px;
  padding: 15px;
  text-align: center;
  border: 1px solid;
  transform: scale(2);
}
#titulo {
  font: bold 26px Verdana, sans-serif;
}
```

Listado 3-73: Escalando la cabecera

En el ejemplo del Listado 3-73, transformamos la cabecera con una escala que duplica su tamaño. A la escala se le pueden asignar números enteros y decimales, y esta escala se calcula por medio de una matriz. Los valores entre 0 y 1 reducen el tamaño del elemento, el valor 1 preserva las proporciones originales, mientras que los valores sobre 1 incrementan las dimensiones del elemento de forma lineal.

Cuando asignamos valores negativos a esta función, se genera un efecto interesante.

```
header {
  margin: 30px;
  padding: 15px;
  text-align: center;
  border: 1px solid;
  transform: scale(1, -1);
}
#titulo {
  font: bold 26px Verdana, sans-serif;
}
```

Listado 3-74: Creando una imagen espejo con la función `scale()`

En el Listado 3-74, se declaran dos parámetros para modificar la escala de la cabecera. Ambos valores preservan las proporciones originales, pero el segundo valor es negativo y, por lo tanto, invierte el elemento en el eje vertical, produciendo una imagen invertida.

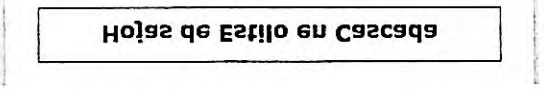

Figura 3-48: Imagen espejo con `scale()`

Además de escalar el elemento, también podemos rotarlo con la función **`rotate()`**. En este caso, los valores negativos cambian la dirección en la cual se rota el elemento. El siguiente ejemplo rota la cabecera 30 grados en el sentido de las agujas del reloj.

```
header {
  margin: 30px;
```

```
  padding: 15px;
  text-align: center;
  border: 1px solid;
  transform: rotate(30deg);
}
#titulo {
  font: bold 26px Verdana, sans-serif;
}
```

Listado 3-75: Rotando la cabecera

Otra función disponible para la propiedad `transform` es `skew()`. Esta función cambia la simetría del elemento en grados y en una o ambas dimensiones.

```
header {
  margin: 30px;
  padding: 15px;
  text-align: center;
  border: 1px solid;
  transform: skew(20deg);
}
#titulo {
  font: bold 26px Verdana, sans-serif;
}
```

Listado 3-76: Inclinando la cabecera

Esta función recibe dos valores, pero a diferencia de otras funciones, cada parámetro solo afecta a una dimensión (los parámetros son independientes). En el Listado 3-76, solo el primer parámetro se declara y, por lo tanto, solo se modifica la dimensión horizontal. Si lo deseamos, podemos usar las funciones adicionales `skewX()` y `skewY()` para lograr el mismo efecto.

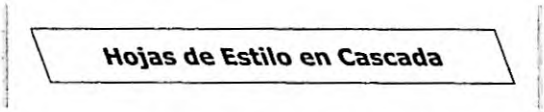

Figura 3-49: Inclinación horizontal

La pantalla de un dispositivo se divide en filas y columnas de píxeles (la mínima unidad visual de la pantalla). Con el propósito de identificar la posición de cada píxel, los ordenadores usan un sistema de coordenadas, donde las filas y columnas de píxeles se cuentan desde la esquina superior izquierda a la esquina inferior derecha de la cuadrícula, comenzando por el valor 0 (los valores se incrementan de izquierda a derecha y de arriba abajo). Por ejemplo, el primer píxel de la esquina superior izquierda de la pantalla se encuentra en la posición 0,0 (columna 0, fila 0), mientras que un píxel que se encuentra 30 píxeles del lado izquierdo de la pantalla y 10 píxeles de la parte superior estará ubicado en la posición 30,10.

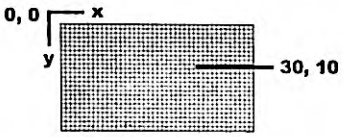

Figura 3-50: Sistema de coordenadas

El punto en la posición 0,0 se llama *origen*, y las líneas de columnas y filas se denominan *ejes* y se identifican con las letras x e y (x para columnas e y para filas), tal como ilustra la Figura 3-50. Usando la propiedad `transform` podemos cambiar la posición de un elemento en esta cuadrícula. La función que tenemos que asignar a la propiedad en este caso es `translate()`.

```
header {
  margin: 30px;
  padding: 15px;
  text-align: center;
  border: 1px solid;
  transform: translate(100px);
}
#titulo {
  font: bold 26px Verdana, sans-serif;
}
```

Listado 3-77: Moviendo la cabecera hacia la derecha

La función `translate()` utiliza el sistema de coordenadas para establecer la posición del elemento, usando su posición actual como referencia. La esquina superior izquierda del elemento se considera que está en la posición `0,0`, por lo que los valores negativos mueven el elemento hacia la izquierda o encima de la posición original, y los valores positivos lo mueven a la derecha y abajo.

En el Listado 3-77, movemos la cabecera a la derecha 100 píxeles de su posición original. Se pueden declarar dos valores en esta función para mover el elemento horizontalmente y verticalmente, o podemos usar las funciones `translateX()` y `translateY()` para declarar los valores de forma independiente.

A veces, puede resultar útil aplicar varias transformaciones a la vez a un mismo elemento. Para combinar transformaciones con la propiedad `transform`, tenemos que declarar las funciones separadas por un espacio.

```
header {
  margin: 30px;
  padding: 15px;
  text-align: center;
  border: 1px solid;
  transform: translateY(100px) rotate(45deg) scaleX(0.3);
}
#titulo {
  font: bold 26px Verdana, sans-serif;
}
```

Listado 3-78: Moviendo, escalando y rotando el elemento con una sola propiedad

IMPORTANTE: cuando combinamos múltiples funciones, debemos considerar que el orden de combinación es importante. Esto se debe a que algunas funciones modifican el punto de origen y el centro del elemento y, por lo tanto, cambian los parámetros sobre los que el resto de las funciones trabajan.

De la misma manera que podemos generar transformaciones en dos dimensiones sobre elementos HTML, también podemos hacerlo en tres dimensiones. Estos tipos de transformaciones se realizan considerando un tercer eje que representa profundidad y se identifica con la letra **z**. Las siguientes son las funciones disponibles para este propósito.

scale3d(x, y, z**)**—Esta función asigna una nueva escala al elemento en un espacio 3D. Acepta tres valores en números decimales para establecer la escala en los ejes **x**, **y** y **z**. Al igual que con transformaciones 2D, el valor 1 preserva la escala original.

rotate3d(x, y, z, ángulo**)**—Esta función rota el elemento en un ángulo y sobre un eje específicos. Los valores para los ejes se deben especificar en números decimales y el ángulo se puede expresar en `deg` (grados), `grad` (gradianes), `rad` (radianes), o `turn` (espiras). Los valores asignados a los ejes determinan un vector de rotación, por lo que los valores no son importantes, pero sí lo es la relación entre los mismos. Por ejemplo, `rotate3d(5, 2, 6, 30deg)` producirá el mismo efecto que `rotate3d(50, 20, 60, 30deg)`, debido a que el vector resultante es el mismo.

translate3d(x, y, z**)**—Esta función mueve el elemento a una nueva posición en el espacio 3D. Acepta tres valores en píxeles para los ejes **x**, **y** y **z**.

perspective(valor**)**—Esta función agrega un efecto de profundidad a la escena incrementando el tamaño del lado del elemento cercano al espectador.

Algunas transformaciones 3D se pueden aplicar directamente al elemento, como hemos hecho con las transformaciones 2D, pero otras requieren que primero declaremos la perspectiva. Por ejemplo, si rotamos el elemento en el eje **y**, un lado del elemento se desplazará hacia adelante y el otro hacia atrás, pero el tamaño de cada lado permanecerá igual y por ello el usuario no verá la transformación. Para lograr un efecto realista, tenemos que declarar la perspectiva con la función `perspective()`.

```
header {
  margin: 30px;
  padding: 15px;
  text-align: center;

  border: 1px solid;
  transform: perspective(500px) rotate3d(0, 1, 0, 45deg);
}
#titulo {
  font: bold 26px Verdana, sans-serif;
}
```

Listado 3-79: Aplicando un efecto tridimensional a la cabecera

La regla del Listado 3-79 primero asigna el valor de la perspectiva y luego rota el elemento 45 grados en el eje **y** (para seleccionar un eje, tenemos que declarar al resto de los ejes con el valor

0). El navegador rota el elemento y, debido a que definimos la perspectiva para la transformación, expande un lado del elemento y reduce el otro para crear la impresión de perspectiva.

Figura 3-51: Efecto 3D con perspectiva

CSS también incluye algunas propiedades que podemos usar para lograr un efecto más realista.

perspective—Esta propiedad trabaja de forma similar a la función `perspective()`, pero opera en el elemento padre. La propiedad crea un contenedor que aplica el efecto de perspectiva a los elementos en su interior.

perspective-origin—Esta propiedad cambia las coordenadas `x` e `y` del espectador. Acepta dos valores en porcentaje, píxeles, o las palabras clave `center`, `left`, `right`, `top` y `bottom`. Los valores por defecto son `50% 50%`.

backface-visibility—Esta propiedad determina si el reverso del elemento será visible o no. Acepta dos valores: `visible` o `hidden`, con el valor `visible` configurado por defecto.

Debido a que en nuestro ejemplo la cabecera del documento es hija directa del cuerpo, tenemos que asignar estas propiedades al elemento `<body>`. El siguiente ejemplo define la perspectiva del cuerpo y luego rota la cabecera con la propiedad `transform`, como hemos hecho anteriormente.

```
body {
  perspective: 500px;
  perspective-origin: 50% 90%;
}
header {
  margin: 30px;
  padding: 15px;
  text-align: center;
  border: 1px solid;
  transform: rotate3d(0, 1, 0, 135deg);
}

#titulo {
  font: bold 26px Verdana, sans-serif;
}
```

Listado 3-80: Declarando un origen diferente para el espectador

El resultado de aplicar la perspectiva al elemento padre es el mismo que si usáramos la función `perspective()` directamente en el elemento que queremos modificar, pero declarando la perspectiva de esta manera podemos usar la propiedad `perspective-origin` y al mismo tiempo cambiar las coordenadas del espectador.

Figura 3-52: Efecto 3D usando la propiedad `perspective`

Hágalo usted mismo: reemplace las reglas en su archivo CSS por las reglas del Listado 3-80 y abra el documento en su navegador. Agregue la propiedad `backface-visibility` con el valor `hidden` a la cabecera para volverla invisible cuando está invertida.

Todas las funciones que hemos estudiado hasta el momento modifican los elementos en el documento, pero una vez que la página web se muestra, permanece igual. Sin embargo, podemos combinar transformaciones y seudoclases para convertir nuestro documento en una aplicación dinámica. La seudoclase que podemos implementar en este caso se llama `:hover`. Las seudoclases, como hemos visto anteriormente, agregan efectos especiales a las reglas. En este caso, la regla con la seudoclase `:hover` se aplica solo cuando el ratón se encuentra sobre el elemento que referencia.

```
header {
  margin: 30px;
  padding: 15px;
  text-align: center;
  border: 1px solid;
}
header:hover {
  transform: rotate(5deg);
}
#titulo {
  font: bold 26px Verdana, sans-serif;
}
```

Listado 3-81: Respondiendo al ratón

En el Listado 3-81, la regla original de la cabecera es la misma, pero ahora se agrega una nueva regla identificada con el selector `header:hover` para aplicar un efecto de transformación cuando el ratón se encuentra sobre el elemento. En consecuencia, cada vez que el usuario mueve el ratón sobre la cabecera, la propiedad `transform` rota el elemento 5 grados, y cuando el puntero se mueve fuera de la caja del elemento, el mismo se rota nuevamente a su posición original. Este código logra una animación básica pero útil usando solo propiedades CSS.

Transiciones

Con la seudoclase `:hover` podemos realizar transformaciones dinámicas. Sin embargo, una animación real requiere una transición entre los dos pasos del proceso. Para este propósito, CSS ofrece las siguientes propiedades.

transition-property—Esta propiedad especifica las propiedades que participan en la transición. Además de los nombres de las propiedades, podemos asignar el valor `all` para indicar que todas las propiedades participarán de la transición.

transition-duration—Esta propiedad especifica la duración de la transición en segundos (`s`).

transition-timing-function—Esta propiedad determina la función que se usa para calcular los valores para la transición. Los valores disponibles son `ease`, `ease-in`, `ease-out`, `ease-in-out`, `linear`, `step-start`, y `step-end`.

transition-delay—Esta propiedad especifica el tiempo que el navegador esperará antes de iniciar la animación.

transition—Esta propiedad nos permite declarar todos los valores de la transición al mismo tiempo.

Implementando estas propiedades le indicamos al navegador que tiene que crear todos los pasos de la animación y generar una transición entre el estado actual del elemento y el especificado por las propiedades. Las siguientes reglas implementan la propiedad `transition` para animar la transformación introducida en el ejemplo anterior.

```
header {
  margin: 30px;
  padding: 15px;
  text-align: center;
  border: 1px solid;
  transition: transform 1s ease-in-out 0s;
}
header:hover {
  transform: rotate(5deg);
}
#titulo {
  font: bold 26px Verdana, sans-serif;
}
```

***Listado 3-82:** Creando una animación con la propiedad* `transition`

La propiedad `transition` puede recibir hasta cuatro parámetros separados por un espacio. El primer valor es la propiedad que se considerará para crear la transición (en nuestro ejemplo, usamos la propiedad `transform`), el segundo parámetro determina la duración de la animación (1 segundo), el tercer parámetro es un valor que determina cómo se llevará a cabo la transición por medio de una curva Bézier, y el último parámetro determina cuántos segundos tarda la animación en comenzar.

Si la transición tiene que considerar los valores de más de una propiedad, tenemos que declarar los nombres de las propiedades separadas por coma. Cuando se tienen que considerar todas las propiedades que se modifican para crear la animación, podemos usar el valor `all` en su lugar.

Hágalo usted mismo: reemplace las reglas en su archivo CSS por las reglas del Listado 3-82 y abra el documento en su navegador. Mueva el puntero del ratón sobre la cabecera para iniciar la animación.

Animaciones

La propiedad `transition` crea una animación básica, pero solo se involucran dos estados en el proceso: el estado inicial determinado por los valores actuales de las propiedades y el estado final, determinado por los nuevos valores. Para crear una animación real, necesitamos declarar más de dos estados, como los fotogramas de una película. CSS incluye las siguientes propiedades para componer animaciones más complejas.

animation-name—Esta propiedad especifica el nombre usado para identificar los pasos de la animación. Se puede usar para configurar varias animaciones al mismo tiempo declarando los nombres separados por coma.

animation-duration—Esta propiedad determina la duración de cada ciclo de la animación. El valor se debe especificar en segundos (por ejemplo, `1s`).

animation-timing-function—Esta propiedad determina cómo se llevará a cabo el proceso de animación por medio de una curva Bézier declarada con los valores `ease`, `linear`, `ease-in`, `ease-out` y `ease-in-out`.

animation-delay—Esta propiedad especifica el tiempo que el navegador esperará antes de iniciar la animación.

animation-iteration-count—Esta propiedad declara la cantidad de veces que se ejecutará la animación. Acepta un número entero o el valor `infinite`, el cual hace que la animación se ejecute por tiempo indefinido. El valor por defecto es `1`.

animation-direction—Esta propiedad declara la dirección de la animación. Acepta cuatro valores: `normal` (por defecto), `reverse`, `alternate`, y `alternate-reverse`. El valor `reverse` invierte la dirección de la animación, mostrando los pasos en la dirección opuesta en la que se han declarado. El valor `alternate` mezcla los ciclos de la animación, reproduciendo los que tienen un índice impar en dirección normal y el resto en dirección invertida. Finalmente, el valor `alternate-reverse` hace lo mismo que `alternate`, pero en sentido inverso.

animation-fill-mode—Esta propiedad define cómo afecta la animación a los estilos del elemento. Acepta los valores `none` (por defecto), `forwards`, `backwards`, y `both`. El valor `forwards` mantiene al elemento con los estilos definidos por las propiedades aplicadas en el último paso de la animación, mientras que `backwards` aplica los estilos del primer paso tan pronto como se define la animación (antes de ser ejecutada). Finalmente, el valor `both` produce ambos efectos.

animation—Esta propiedad nos permite definir todos los valores de la animación al mismo tiempo.

Estas propiedades configuran la animación, pero los pasos se declaran por medio de la regla `@keyframes`. Esta regla se debe identificar con el nombre usado para configurar la animación, y debe incluir la lista de propiedades que queremos modificar en cada paso. La posición de cada paso de la animación se determina con un valor en porcentaje, donde 0 % corresponde al primer fotograma o al comienzo de la animación, y 100 % corresponde al final.

```
header {
  margin: 30px;
  padding: 15px;
```

```
  text-align: center;
  border: 1px solid;
  animation: mianimacion 1s ease-in-out 0s infinite normal none;
}
@keyframes mianimacion {
  0% {
    background: #FFFFFF;
  }
  50% {
    background: #FF0000;
  }
  100% {
    background: #FFFFFF;
  }
}
#titulo {
  font: bold 26px Verdana, sans-serif;
}
```

Listado 3-83: Creando una animación compleja

Las reglas del Listado 3-83 crean una animación que cambia los colores del fondo de la cabecera de rojo a blanco. La animación se ha definido mediante la propiedad **animation** con una duración de 1 segundo, y configurado para ejecutarse una y otra vez con el valor **infinite**. La propiedad también asigna el nombre **mianimacion** a la animación para poder configurar luego los pasos con la regla **@keyframes**.

Las propiedades indican en cada paso cómo será afectará al elemento. En este caso, declaramos tres pasos, uno al 0 %, otro al 50 %, y un tercero al 100 %. Cuando se inicia la animación, el navegador asigna al elemento los estilos definidos al 0 % y luego cambia los valores de las propiedades gradualmente hasta llegar a los valores definidos al 50 %. El proceso se repite desde este valor hasta el valor final asignado a la propiedad en el último paso (100 %).

En este ejemplo, definimos el estado inicial de la animación al 0 % y el estado final al 100 %, pero también podemos iniciar la animación en cualquier otro punto y declarar todos los pasos que necesitemos, como en el siguiente ejemplo.

```
header {
  margin: 30px;
  padding: 15px;
  text-align: center;
  border: 1px solid;
  animation: mianimacion 1s ease-in-out 0s infinite normal none;
}
@keyframes mianimacion {
  20% {
    background: #FFFFFF;
  }
  35% {
    transform: scale(0.5);
    background: #FFFF00;
  }
  50% {
    transform: scale(1.5);
```

```
    background: #FF0000;
  }
  65% {
    transform: scale(0.5);
    background: #FFFF00;
  }
  80% {
    background: #FFFFFF;
  }
}
#titulo {
  font: bold 26px Verdana, sans-serif;
}
```

Listado 3-84: Declarando más pasos para nuestra animación

En el Listado 3-84, la animación comienza al 20 % y termina al 80 %, e incluye un total de cinco pasos. Cada paso de la animación modifica dos propiedades que incrementan el tamaño del elemento y cambian el color de fondo, excepto el primer paso y el último que solo cambian el color para lograr un efecto de rebote.

Capítulo 4
Diseño web

4.1 Cajas

Como hemos mencionado en el capítulo anterior, los navegadores crean una caja virtual alrededor de cada elemento para determinar el área que ocupan. Para organizar estas cajas en la pantalla, los elementos se clasifican en dos tipos básicos: Block (bloque) e Inline (en línea). La diferencia principal entre estos dos tipos es que los elementos Block tienen un tamaño personalizado y generan saltos de línea, mientras que los elementos Inline tienen un tamaño determinado por su contenido y no generan saltos de línea. Debido a sus características, los elementos Block se colocan de uno en uno en las distintas líneas, y los elementos Inline se colocan uno al lado del otro en la misma línea, a menos que no haya suficiente espacio horizontal disponible, como lo ilustra la Figura 4-1.

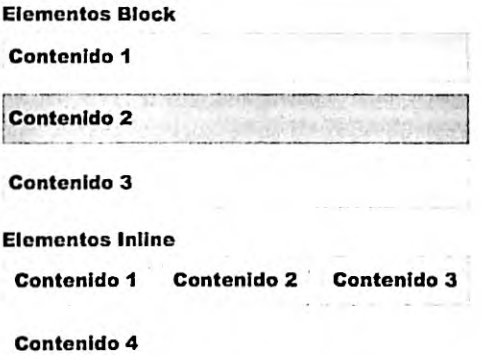

Figura 4-1: Elementos *Block e Inline*

Debido a sus características, los elementos Block son apropiados para crear columnas y secciones en una página web, mientras que los elementos Inline son adecuados para representar contenido. Esta es la razón por la que los elementos que definen la estructura de un documento, como `<section>`, `<nav>`, `<header>`, `<footer>`, o `<div>`, se declaran como elementos Block por defecto, y otros como `<span>`, `<strong>`, o `<em>`, que representan el contenido de esos elementos, se declaran como elementos Inline.

Display

El que un elemento sea del tipo Block o Inline lo determina el navegador, pero podemos cambiar esta condición desde CSS con la siguiente propiedad.

display—Esta propiedad define el tipo de caja usado para presentar el elemento en pantalla. Existen varios valores disponibles para esta propiedad. Los más utilizados son

none (elimina el elemento), **block** (muestra el elemento en una nueva línea y con un tamaño personalizado), **inline** (muestra el elemento en la misma línea), e **inline-block** (muestra el elemento en la misma línea y con un tamaño personalizado).

Los elementos estructurales se configuran por defecto con el valor **block**, mientras que los elementos que representan el contenido normalmente se configuran como **inline**. Si queremos modificar el tipo de elemento, solo tenemos que asignar la propiedad **display** con un nuevo valor. Así, las antiguas versiones de navegadores no reconocen los nuevos elementos incorporados por HTML5 y los consideran como elementos Inline por defecto. Si queremos asegurarnos de que estos elementos se interpreten como elementos Block en todos los navegadores, podemos declarar la siguiente regla en nuestras hojas de estilo.

```
header, section, main, footer, aside, nav, article, figure, figcaption
{
  display: block;
}
```

Listado 4-1: Definiendo los elementos HTML5 como elementos Block

La propiedad **display** cuenta con otros valores además de **block** e **inline**. Por ejemplo, el valor **none** oculta el elemento. Cuando este valor se asigna a un elemento, el documento se presenta como si el elemento no existiera. Es útil cuando queremos cambiar el documento dinámicamente desde JavaScript o cuando usamos diseño web adaptable para diseñar nuestro sitio web, como veremos en próximos capítulos.

Lo básico: el valor **none** para la propiedad **display** elimina el elemento del documento. Si lo que queremos es volver al elemento invisible, podemos usar otra propiedad CSS llamada **visibility**. Esta propiedad acepta los valores **visible** y **hidden**. Cuando el valor **hidden** se asigna a la propiedad, se generan el elemento y su contenido (ocupan un espacio en la pantalla), pero no se muestran al usuario.

Otro valor disponible para la propiedad **display** es **inline-block**. Los elementos Block presentan dos características, una es que producen un salto de línea, por lo que el siguiente elemento se muestra en una nueva línea, y la otra es que pueden adoptar un tamaño personalizado. Esta es la razón por la que las propiedades **width** y **height** estudiadas anteriormente solo trabajan con elementos Block. Si asignamos estas propiedades a un elemento Inline como ****, no ocurre nada. Pero la propiedad **display** ofrece el valor **inline-block** para definir un elemento Inline que puede adoptar un tamaño personalizado. Esto significa que los elementos Inline-Block se posicionarán uno al lado del otro en la misma fila, pero con el tamaño que queramos.

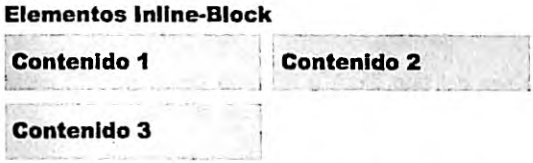

Figura 4-2: Elementos Inline-Block

Los elementos Inline-Block nos permiten crear secciones en nuestra página web del tamaño que deseemos y ubicarlas en la misma línea si lo necesitamos. Por ejemplo, si tenemos dos elementos Block que se deben mostrar uno al lado del otro, como los elementos `<section>` y `<aside>` del documento desarrollado en el Capítulo 2, podemos declararlos como elementos Inline-Block.

Aunque puede resultar tentador usar elementos Inline-Block para diseñar todas las columnas y secciones de nuestras páginas web, CSS incluye mejores propiedades para este propósito. Estas propiedades son parte de lo que llamamos *modelo de cajas*, un conjunto de reglas que determinan cómo se van a mostrar las cajas en pantalla, el espacio que ocupan, y cómo se organizan en la página considerando el espacio disponible.

Actualmente hay varios modelos de cajas disponibles, con el modelo de caja tradicional y el modelo de caja flexible considerados como estándar. Para aprender a diseñar nuestras páginas web debemos entender cómo funcionan estos dos modelos.

4.2 Modelo de caja tradicional

Como ya mencionamos, los elementos Block se colocan unos sobre otros y los elementos Inline se posicionan de izquierda a derecha en la misma línea. El modelo de caja tradicional establece que los elementos pueden flotar a cada lado de la ventana y compartir espacio en la misma línea con otros elementos, sin importar su tipo. Por ejemplo, si tenemos dos elementos Block que representan columnas en el diseño, podemos posicionar una columna a la izquierda y la otra columna a la derecha haciendo que los elementos floten hacia el lado que queremos. Las siguientes son las propiedades que ofrece CSS para este propósito.

float—Esta propiedad permite a un elemento flotar hacia un lado u otro, y ocupar el espacio disponible, incluso cuando tiene que compartir la línea con otro elemento. Los valores disponibles son **none** (el elemento no flota), **`left`** (el elemento flota hacia la izquierda) y **`right`** (el elemento flota hacia la derecha).

clear—Esta propiedad restaura el flujo normal del documento, y no permite que el elemento siga flotando hacia la izquierda, la derecha o ambos lados. Los valores disponibles son **none**, **`left`**, **`right`**, y **both** (ambos).

La propiedad **`float`** hace que el elemento flote a un lado u otro en el espacio disponible. Cuando se aplica esta propiedad, los elementos no siguen el flujo normal del documento, se desplazan a la izquierda o a la derecha del espacio disponible, respondiendo al valor de la propiedad **`float`** y hasta que especifiquemos lo contrario con la propiedad **`clear`**.

Contenido flotante

Las propiedades **`float`** y **`clear`** se usaron originalmente para hacer que el contenido flote alrededor de un elemento. Por ejemplo, si queremos que un texto se muestre al lado de una imagen, podemos hacer flotar la imagen hacia la izquierda o la derecha, y el texto compartirá con la imagen el espacio disponible en la misma línea.

```
<!DOCTYPE html>
<html lang="es">
```

```
<head>
  <title>Este texto es el título del documento</title>
  <meta charset="utf-8">
  <meta name="description" content="Este es un documento HTML5">
  <meta name="keywords" content="HTML, CSS, JavaScript">
  <link rel="stylesheet" href="misestilos.css">
</head>
<body>
  <section>
    <img src="miimagen.jpg" width="120">
    <p>HTML, sigla en inglés de HyperText Markup Language (lenguaje de
marcas de hipertexto), hace referencia al lenguaje de marcado para la
elaboración de páginas web. Se considera el lenguaje web más importante
siendo su invención crucial en la aparición, desarrollo y expansión de
la World Wide Web (WWW).</p>
  </section>
  <footer>
    <p>Publicado por Wikipedia</p>
  </footer>
</body>
</html>
```

***Listado 4-2:** Probando la propiedad* `float`

El documento del Listado 4-2 incluye una sección con una imagen y un párrafo. Si abrimos este documento usando los estilos por defecto, el elemento **<p>** se muestra debajo del elemento **<img>**.

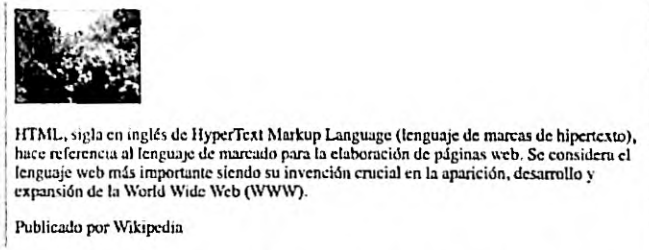

***Figura 4-3:** Imagen y párrafo con estilos por defecto*

Si queremos mostrar el texto al lado de la imagen, podemos hacer que el elemento **<img>** flote hacia la izquierda o la derecha.

```
section img {
  float: left;
  margin: 0px 10px;
}
```

***Listado 4-3:** Flotando la imagen hacia la izquierda*

Cuando un elemento flota hacia un lado, los siguientes elementos flotan a su alrededor ocupando el espacio disponible.

HTML, sigla en inglés de HyperText Markup Language (lenguaje de marcas de hipertexto), hace referencia al lenguaje de marcado para la elaboración de páginas web. Se considera el lenguaje web más importante siendo su invención crucial en la aparición, desarrollo y expansión de la World Wide Web (WWW).

Publicado por Wikipedia

Figura 4-4: Imagen flota hacia la izquierda

Si en lugar del valor `left` asignamos el valor `right` a la propiedad `float`, la imagen flota hacia la derecha y el texto la sigue, ocupando el espacio disponible del lado izquierdo.

HTML, sigla en inglés de HyperText Markup Language (lenguaje de marcas de hipertexto), hace referencia al lenguaje de marcado para la elaboración de páginas web. Se considera el lenguaje web más importante siendo su invención crucial en la aparición, desarrollo y expansión de la World Wide Web (WWW).

Publicado por Wikipedia

Figura 4-5: Imagen flota hacia la derecha

Hágalo usted mismo: cree un nuevo archivo HTML con el documento del Listado 4-2 y un archivo CSS con el nombre misestilos.css y la regla del Listado 4-3. Recuerde incluir la imagen miimagen.jpg en el mismo directorio (la imagen está disponible en nuestro sitio web). Abra el documento en su navegador. Debería ver algo similar a lo que muestra la Figura 4-4. Asigne el valor `right` a la propiedad `float` y actualice el documento en su navegador. La imagen se debería mover hacia la derecha, como muestra la Figura 4-5.

Los navegadores no pueden calcular el tamaño de un contenedor a partir del tamaño de elementos flotantes, por lo que si el elemento afectado por la propiedad `float` es más alto que el resto de los elementos de la misma línea, desbordará al contenedor. Por ejemplo, la siguiente figura ilustra lo que ocurre si eliminamos el atributo `width` del elemento `<img>` en nuestro ejemplo y dejamos que el navegador muestre la imagen en su tamaño original (en este ejemplo, asignamos un fondo gris al elemento `<section>` para poder identificar el área que ocupa).

HTML, sigla en inglés de HyperText Markup Language (lenguaje de marcas de hipertexto), hace referencia al lenguaje de marcado para la elaboración de páginas web. Se considera el lenguaje web más importante siendo su invención crucial en la aparición, desarrollo y expansión de la World Wide Web (WWW).

Publicado por Wikipedia

Figura 4-6: La imagen es más alta que su contenedor

El navegador estima el tamaño del contenedor de acuerdo al tamaño del texto y, por lo tanto, la imagen se sitúa fuera de los límites del contenedor. Debido a que la imagen flota

hacia la izquierda y se extiende fuera del elemento **<section>**, el contenido del siguiente elemento sigue flotando hasta ocupar el espacio dejado por la imagen y de este modo obtenemos el resultado mostrado en la Figura 4-6 (el elemento **<footer>** se muestra al lado derecho de la imagen en lugar de estar debajo de la misma).

Una forma de asegurarnos de que el contenedor es lo suficientemente alto como para contener los elementos flotantes, es asignándole la propiedad **overflow** con el valor **auto**. Esto fuerza al navegador a calcular el tamaño del contenedor considerando todos los elementos en su interior.

```
section {
  background-color: #CCCCCC;
  overflow: auto;
}
section img {
  float: left;
  margin: 0px 10px;
}
```

***Listado 4-4:** Recuperando el flujo normal del documento con la propiedad* **overflow**

La propiedad **overflow** no deja que el contenido desborde a su contenedor extendiendo el tamaño del contenedor o incluyendo barras de desplazamiento para permitir al usuario ver todo el contenido si el tamaño del contenedor no se puede cambiar.

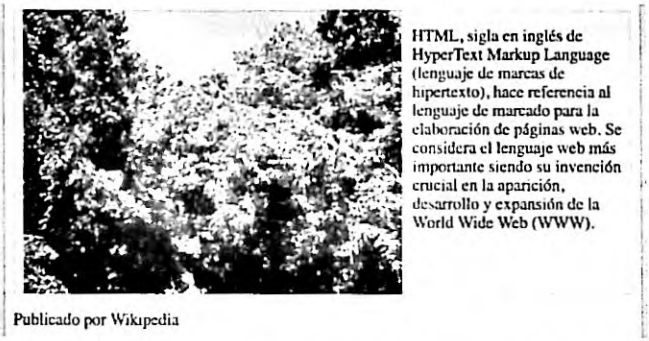

***Figura 4-7:** La imagen ya no desborda a su contenedor*

Hágalo usted mismo: elimine el atributo **width** del elemento **<img>** en el documento HTML del Listado 4-2. Reemplace las reglas en su archivo CSS por el código del Listado 4-4. Abra el documento en su navegador. Debería ver algo parecido a la Figura 4-7.

Otra alternativa para normalizar el flujo del documento es declarar la propiedad **clear** en el elemento que se encuentra a continuación del elemento flotante dentro del contenedor. Debido a que no siempre tenemos un elemento hermano después de un elemento flotante que podamos usar para prevenir que los elementos sigan flotando, esta técnica requiere que agreguemos un elemento adicional al documento. Por ejemplo, podemos incluir un elemento

** ** debajo del elemento **<p>** dentro del elemento **<section>** de nuestro documento para impedir que los siguientes elementos continúen flotando hacia los lados.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Este texto es el título del documento</title>
  <meta charset="utf-8">
  <meta name="description" content="Este es un documento HTML5">
  <meta name="keywords" content="HTML, CSS, JavaScript">
  <link rel="stylesheet" href="misestilos.css">
</head>
<body>
  <section>
    <img src="miimagen.jpg">
    <p>HTML, sigla en inglés de HyperText Markup Language (lenguaje de
marcas de hipertexto), hace referencia al lenguaje de marcado para la
elaboración de páginas web. Se considera el lenguaje web más importante
siendo su invención crucial en la aparición, desarrollo y expansión de
la World Wide Web (WWW).</p>
    <div class="clearfloat"></div>
  </section>
  <footer>
    <p>Publicado por Wikipedia</p>
  </footer>
</body>
</html>
```

Listado 4-5: Agregando un elemento vacía para aplicar la propiedad `clear`

En este caso, debemos asignar la propiedad **clear** al elemento adicional. La propiedad ofrece valores para restaurar el flujo de un lado o ambos. Debido a que normalmente necesitamos restaurar el flujo normal del documento en ambos lados, el valor preferido es **both**.

```
section {
  background-color: #CCCCCC;
}
section img {
  float: left;
  margin: 0px 10px;
}
.clearfloat {
  clear: both;
}
```

Listado 4-6: Restaurando el flujo normal del documento con la propiedad `clear`

Con las reglas del Listado 4-6, el elemento ** ** restaura el flujo normal del documento, permitiendo al navegador calcular el tamaño del contenedor a partir de su contenido, con lo que obtenemos un resultado similar al que vimos en la Figura 4-7.

Hágalo usted mismo: actualice su archivo HTML con el documento del Listado 4-5 y reemplace las reglas en su archivo CSS por el código del Listado 4-6. Abra el documento en su navegador. Debería ver algo similar a la Figura 4-7.

Cajas flotantes

La propiedad **clear** no afecta a otros aspectos del elemento y no agrega barras de desplazamiento como la propiedad **overflow**. Por lo tanto, es la que se implementa en el modelo de caja tradicional para organizar la estructura de un documento. Con las propiedades **float** y **clear** podemos controlar con precisión dónde se mostrarán los elementos en pantalla y diseñar nuestras páginas web. Por ejemplo, el siguiente documento incluye un elemento **<section>** con cuatro elementos ** ** que debemos hacer flotar a un lado para convertirlos en columnas dentro de la página, y un elemento ** ** al final que usaremos para recuperar el flujo normal del documento.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Este texto es el título del documento</title>
  <meta charset="utf-8">
  <meta name="description" content="Este es un documento HTML5">
  <meta name="keywords" content="HTML, CSS, JavaScript">
  <link rel="stylesheet" href="misestilos.css">
</head>
<body>
  <section id="cajapadre">
    <div id="caja-1">Caja 1</div>
    <div id="caja-2">Caja 2</div>
    <div id="caja-3">Caja 3</div>
    <div id="caja-4">Caja 4</div>
    <div class="restaurar"></div>
  </section>
</body>
</html>
```

***Listado 4-7:** Creando un documento para probar la propiedad* `float`

Como ha ocurrido con otros documentos anteriormente, si abrimos este documento sin asignar estilos personalizados, el contenido de sus elementos se muestra de arriba abajo, siguiendo el flujo del documento por defecto.

***Figura 4-8:** Flujo normal del documento*

Debido a que queremos que estas cajas se ubiquen una al lado de la otra representando columnas de nuestra página web, tenemos que asignarles un tamaño fijo y hacerlas flotar a un lado o al otro. El tamaño se determina mediante las propiedades **width** y **height**, y el modo

en el que flotan lo determina la propiedad **float**, pero el valor asignado a esta última propiedad depende de lo que queremos lograr. Si flotamos las cajas hacia la izquierda, estas se alinearán de izquierda a derecha, y si las hacemos flotar hacia la derecha, harán lo mismo pero de derecha a izquierda. Por ejemplo, si las hacemos flotar a la izquierda, **caja-1** se colocará en primer lugar en el lado izquierdo de la caja padre, y luego el resto de las cajas se ubicarán a su derecha en el orden en el que se han declarado en el código (las cajas flotan hacia la izquierda hasta que colisionan con el límite del contenedor o la caja anterior).

```
#cajapadre {
  width: 600px;
  border: 1px solid;
}
#caja-1 {
  float: left;
  width: 140px;
  height: 50px;
  margin: 5px;
  background-color: #AAAAAA;
}
#caja-2 {
  float: left;
  width: 140px;
  height: 50px;
  margin: 5px;
  background-color: #CCCCCC;
}
#caja-3 {
  float: left;
  width: 140px;
  height: 50px;
  margin: 5px;
  background-color: #AAAAAA;
}
#caja-4 {
  float: left;
  width: 140px;
  height: 50px;
  margin: 5px;
  background-color: #CCCCCC;
}
.restaurar {
  clear: both;
}
```

Listado 4-8: Flotando cajas a la izquierda

En el ejemplo del Listado 4-8, asignamos un ancho de 600 píxeles al elemento **<section>** y un tamaño de 140 píxeles por 50 píxeles a cada caja. Siguiendo el flujo normal, estas cajas se apilarían una encima de la otra, pero debido a que les asignamos la propiedad **float** con el valor **left**, flotan hacia la izquierda hasta que se encuentran con el límite del contenedor u otra caja, llenando el espacio disponible en la misma línea.

Figura 4-9: Cajas organizadas con la propiedad `float`

Como mencionamos anteriormente, cuando el contenido de un elemento flota, el elemento padre no puede calcular su propia altura desde la altura de su contenido. Esta es la razón por la que, cada vez que tenemos elementos que flotan, debemos recuperar el flujo normal en el elemento siguiente con la propiedad `clear`.

Hágalo usted mismo: cree un nuevo archivo HTML con el código del Listado 4-7 y un nuevo archivo CSS llamado misestilos.css con las reglas del Listado 4-8. Abra el documento en su navegador. Debería ver algo similar a la Figura 4-9.

En el último ejemplo, nos aseguramos de que el elemento padre es suficientemente ancho como para contener todas las cajas y, por lo tanto, todas se muestran en la misma línea, pero si el contenedor no tiene espacio suficiente, los elementos que no se pueden ubicar en la misma línea se moverán a una nueva. La siguiente regla reduce el tamaño del elemento `<section>` a 500 píxeles.

```
#cajapadre {
  width: 500px;
  border: 1px solid;
}
```

Listado 4-9: Reduciendo el tamaño del contenedor

Después de aplicar esta regla a nuestro documento, la última caja no encuentra espacio suficiente al lado derecho de la tercera caja y, por lo tanto, flota hacia el lado izquierdo del contenedor en una nueva línea.

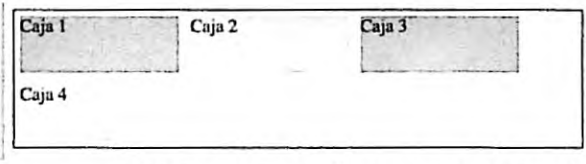

Figura 4-10: Las cajas llenan el espacio disponible

Por otro lado, si tenemos más espacio disponible en el contenedor del que necesitan las cajas, el espacio restante se ubica a los lados (izquierdo o derecho, dependiendo del valor de la propiedad `float`). Si queremos ubicar el espacio restante en medio de las cajas, podemos hacer flotar algunas cajas hacia la izquierda y otras hacia la derecha. Por ejemplo, las siguientes reglas asignan un tamaño de 120 píxeles a cada caja, dejando un espacio vacío de 80 píxeles, pero como las dos últimas cajas flotan a la derecha en lugar de a la izquierda, el espacio libre se ubica en el medio del contenedor, no a los lados.

```
#cajapadre {
  width: 600px;
  border: 1px solid;
}
#caja-1 {
  float: left;
  width: 120px;
  height: 50px;
  margin: 5px;
  background-color: #AAAAAA;
}
#caja-2 {
  float: left;
  width: 120px;
  height: 50px;
  margin: 5px;
  background-color: #CCCCCC;
}
#caja-3 {
  float: right;
  width: 120px;
  height: 50px;
  margin: 5px;
  background-color: #AAAAAA;
}
#caja-4 {
  float: right;
  width: 120px;
  height: 50px;
  margin: 5px;
  background-color: #CCCCCC;
}
.restaurar {
  clear: both;
}
```

***Listado 4-10:** Flotando las cajas a izquierda y derecha*

Los elementos **caja-1** y **caja-2** flotan a la izquierda, lo que significa que **caja-1** se ubicará en el lado izquierdo del contenedor y **caja-2** se ubicará en el lado derecho de **caja-1**, pero **caja-3** y **caja-4** flotan a la derecha, por lo que van a estar ubicadas en el lado derecho del contenedor, dejando un espacio en el medio.

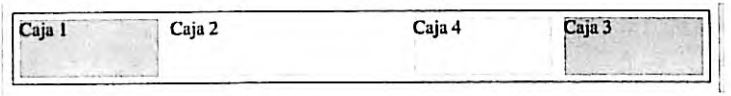

***Figura 4-11:** Espacio libre entre las cajas*

Debido al orden en el que se han declarado los elementos en el código, el elemento **caja-4** se ubica en el lado izquierdo del elemento **caja-3**. El navegador procesa los elementos en el orden en el que se han declarado en el documento; por lo tanto, cuando se procesa el

elemento **caja-3**, se mueve a la derecha hasta que alcanza el límite derecho del contenedor, pero cuando se procesa el elemento **caja-4**, no se puede mover hacia el lado derecho del contenedor porque el elemento **caja-3** ya ocupa ese lugar y, por lo tanto, se ubica en el lado izquierdo de **caja-3**. Si queremos que las cajas se muestren en orden, tenemos que modificar el documento del Listado 4-7 y mover el elemento ** ** identificado con el nombre **caja-4** sobre el elemento ** ** identificado con el nombre **caja-3**.

Posicionamiento absoluto

La posición de un elemento puede ser relativa o absoluta. Con una posición es relativa, las cajas se colocan una después de la otra en el espacio designado por el contenedor. Si el espacio no es suficiente o los elementos no son flotantes, las cajas se colocan en una nueva línea. Este es el modo de posicionamiento por defecto, pero existe otro modo llamado *posicionamiento absoluto*. El posicionamiento absoluto nos permite especificar las coordenadas exactas en las que queremos posicionar cada elemento. Las siguientes son las propiedades disponibles para este propósito.

position—Esta propiedad define el tipo de posicionamiento usado para colocar un elemento. Los valores disponibles son **static** (se posiciona de acuerdo con el flujo normal del documento), **relative** (se posiciona según la posición original del elemento), **absolute** (se posiciona con una posición absoluta relativa al contenedor del elemento), y **fixed** (se posiciona con una posición absoluta relativa a la ventana del navegador).

top—Esta propiedad especifica la distancia entre el margen superior del elemento y el margen superior de su contenedor.

bottom—Esta propiedad especifica la distancia entre el margen inferior del elemento y el margen inferior de su contenedor.

left—Esta propiedad especifica la distancia entre el margen izquierdo del elemento y el margen izquierdo de su contenedor.

right—Esta propiedad especifica la distancia entre el margen derecho del elemento y el margen derecho de su contenedor.

Las propiedades **top**, **bottom**, **left**, y **right** se aplican en ambos tipos de posicionamiento, relativo o absoluto, pero trabajan de diferentes maneras. Cuando el elemento se ubica con posicionamiento relativo, el elemento se desplaza pero el diseño no se modifica. Con posicionamiento absoluto, el elemento se elimina del diseño, por lo que el resto de los elementos también se desplazan para ocupar el nuevo espacio libre.

El siguiente ejemplo usa posicionamiento relativo para desplazar la primera caja de nuestro ejemplo 25 píxeles hacia abajo.

```
#caja-1 {
  position: relative;
  top: 25px;
  float: left;
  width: 140px;
  height: 50px;
  margin: 5px;
```

```
  background-color: #AAAAAA;
}
```

Listado 4-11: Especificando la posición relativa de un elemento

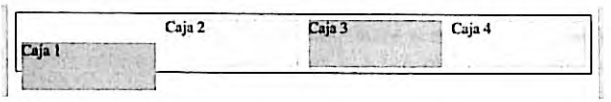

Figura 4-12: Posición relativa

Hágalo usted mismo: reemplace la regla `caja-1` en su hoja de estilo CSS por la regla del Listado 4-11. En este ejemplo, solo desplazamos la primera caja, pero no tocamos las reglas para el resto de las cajas. Agregue otras propiedades como `left` o `right` para ver cómo afectan a la posición del elemento.

Otra alternativa es usar posicionamiento absoluto. En este caso, el elemento se elimina del diseño, por lo que el resto de los elementos se ven afectados por la regla. Cuando usamos posicionamiento absoluto, también tenemos que considerar que el elemento se ubicará con respecto a la ventana del navegador, a menos que declaremos la posición de su elemento padre. Por lo tanto, si queremos especificar una posición absoluta para un elemento basada en la posición de su elemento padre, también tenemos que declarar la propiedad `position` para el padre. En el siguiente ejemplo, declaramos una posición relativa para el elemento `cajapadre` y una posición absoluta de 25 píxeles desde la parte superior para la `caja-1`, lo que hará que se ubique en una posición más baja que el resto de las cajas.

```
#cajapadre {
  position: relative;
  width: 600px;
  border: 1px solid;
}
#caja-1 {
  position: absolute;
  top: 25px;

  float: left;
  width: 140px;
  height: 50px;
  margin: 5px;
  background-color: #AAAAAA;
}
```

Listado 4-12: Especificando la posición absoluta de un elemento

Debido a que el posicionamiento absoluto elimina al elemento del diseño del documento, el resto de las cajas se mueven a la izquierda para ocupar el espacio vacío que ha dejado `caja-1`.

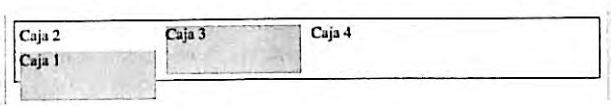

Figura 4-13: Posición absoluta

Hágalo usted mismo: reemplace las reglas **#cajapadre** y **#caja-1** en su hoja de estilo CSS con las reglas del Listado 4-12 y abra el documento en su navegador. Debería ver algo parecido a la Figura 4-13.

El orden de los elementos del código no solo determina la ubicación de las cajas en la página, sino también qué caja va a estar por encima de las demás cuando se superponen. Debido a que en nuestro ejemplo **caja-1** se ha declarado primero en el código, se dibuja sobre **caja-2**, pero CSS ofrece la siguiente propiedad para cambiar este comportamiento.

z-index—Esta propiedad define un índice que determina la posición del elemento en el eje z. El elemento con el índice más alto se dibujará sobre el elemento con el índice más bajo.

Por ejemplo, podemos mover el elemento **caja-1** debajo del elemento **caja-2** y sobre el elemento **cajapadre** asignando índices negativos a **caja-1** y **cajapadre**.

```
#cajapadre {
  position: relative;
  width: 600px;
  border: 1px solid;
  z-index: -2;
}
#caja-1 {
  position: absolute;
  top: 25px;
  z-index: -1;

  float: left;
  width: 140px;
  height: 50px;
  margin: 5px;
  background-color: #AAAAAA;
}
```

***Listado 4-13:** Especificando el índice z*

Los índices negativos se consideran más bajos que los índices asignados por defecto. En el Listado 4-13, asignamos el valor -1 al elemento **caja-1** y el valor -2 al elemento **cajapadre**. Esto mueve **caja-1** debajo de **caja-2**, pero mantiene **caja-1** sobre **cajapadre**, porque su índice es más alto.

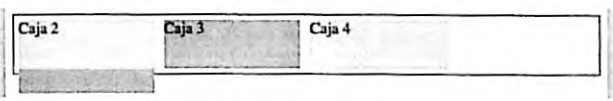

***Figura 4-14:** Índice z*

Cuando usamos posicionamiento relativo y absoluto, el diseño del documento se modifica y, por lo tanto, esta técnica no se usa para organizar los elementos en pantalla, sino más bien para crear efectos en los cuales los elementos ocultos se muestran respondiendo a acciones del usuario, como cuando necesitamos crear menús desplegables o listas desplazables que revelan información adicional. Por ejemplo, podemos mostrar una caja con el título de una imagen cuando el usuario mueve el ratón sobre ella.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Este texto es el título del documento</title>
  <meta charset="utf-8">
  <meta name="description" content="Este es un documento HTML5">
  <meta name="keywords" content="HTML, CSS, JavaScript">
  <link rel="stylesheet" href="misestilos.css">
</head>
<body>
  <section id="cajapadre">
    <img src="miimagen.jpg" width="365" height="274">
    <div id="contenedor">
      <div id="contenedorsuperior"></div>
      <div id="contenedorinferior">
        <span><strong>Este es mi patio</strong></span>
      </div>
    </div>
  </section>
</body>
</html>
```

Listado 4-14: Mostrando una etiqueta desplegable con información adicional

El documento del Listado 4-14 incluye una sección que contiene una imagen. Debajo del elemento **<img>** se encuentra un elemento ** ** identificado con el nombre **contenedor** y dentro de este elemento tenemos dos elementos ** ** más, uno vacío y el otro con un título. El propósito de este código es cubrir la imagen con una caja que, cuando se mueve hacia arriba, revela otra caja en la parte inferior con el título de la imagen. La Figura 4-15 ilustra la estructura generada por estos elementos.

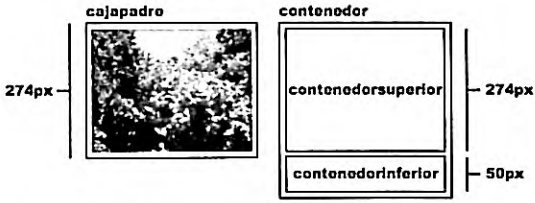

Figura 4-15: Estructura para presentar una etiqueta desplegable con información adicional

La imagen y el contenedor se encuentran dentro del elemento **cajapadre**, pero el contenedor tiene que colocarse encima de la imagen, de modo que se pueda mover para revelar el elemento **contenedorinferior** con el título cuando el usuario posiciona el ratón sobre la imagen. Por esta razón, tenemos que asignar una posición absoluta a este elemento.

```
#cajapadre {
  position: relative;
  width: 365px;
  height: 274px;
  overflow: hidden;
}
```

```
#contenedor {
  position: absolute;
  top: 0px;
  width: 365px;
  height: 324px;
}
#contenedorsuperior {
  width: 365px;
  height: 274px;
}
#contenedorinferior {
  width: 365px;
  height: 35px;
  padding-top: 15px;
  background-color: rgba(200, 200, 200, 0.8);
  text-align: center;
}
#contenedor:hover {
  top: -50px;
}
```

Listado 4-15: Configurando las cajas

La imagen que usamos en este ejemplo tiene un tamaño de 365 píxeles de ancho por 274 píxeles de alto, por lo que tenemos que especificar este tamaño para el elemento **cajapadre**. El contenedor, por otro lado, tiene que ser más alto porque debe contener el elemento **contenedorsuperior** y el elemento **contenedorinferior** que se revela cuando se mueve hacia arriba. Debido a que la caja con el título tiene una altura de 50 píxeles (35 píxeles de altura y 15 píxeles de relleno), le asignamos a la caja contenedora una altura de 324 píxeles (274 + 50).

La razón por la que ubicamos un contenedor por encima de la imagen es porque tenemos que reaccionar cuando el usuario mueve el ratón sobre la imagen. CSS únicamente nos permite hacerlo con la seudoclase **:hover**, como hemos visto anteriormente (vea los Listados 3-81 y 3-82). El problema de esta seudoclase es que solo nos permite modificar el elemento al que se ha aplicado. Usándola con la imagen, solo podríamos modificar la imagen misma, pero aplicándola al contenedor, podemos cambiar el valor de su propiedad **top** para moverlo hacia arriba y revelar el elemento **contenedorinferior**. La regla al final del Listado 4-15 realiza esta tarea. Cuando el usuario mueve el ratón sobre la imagen, la regla **#contenedor:hover** asigna un valor de -50 píxeles a la propiedad **top** del elemento **contenedor**, moviendo el elemento y su contenido hacia arriba, para revelar el título de la imagen.

Figura 4-16: Etiqueta sobre la imagen

El elemento **contenedorinferior** se muestra tan pronto como el ratón se mueve sobre la imagen. Esto se debe a que no declaramos ninguna transición para la propiedad **top**. El valor va de 0px a -50px instantáneamente, por lo que no vemos ninguna transición en el proceso. Para declarar pasos intermedios y crear una animación, tenemos que agregar la propiedad **transition** al elemento **contenedor**.

```
#contenedor {
  position: absolute;
  top: 0px;
  width: 365px;
  height: 324px;
  transition: top 0.5s ease-in-out 0s;
}
```

Listado 4-16: Animando la etiqueta

Hágalo usted mismo: cree un nuevo archivo HTML con el código del Listado 4-14 y un archivo CSS llamado misestilos.css con el código del Listado 4-15. Descargue la imagen miimagen.jpg desde nuestro sitio web. Abra el documento en su navegador y mueva el ratón sobre la imagen. Debería ver la etiqueta con el título de la imagen aparecer y desaparecer en la parte inferior de la imagen (Figura 4-16). Reemplace la regla **#contenedor** de su hoja de estilo con la regla del Listado 4-16. Ahora la etiqueta debería ser animada.

Columnas

Además de las propiedades que hemos estudiado para organizar las caja en pantalla, CSS también incluye un grupo de propiedades para facilitar la creación de columnas.

column-count—Esta propiedad especifica el número de columnas que el navegador tiene que generar para distribuir el contenido del elemento.

column-width—Esta propiedad declara el ancho de las columnas. El valor se puede declarar como **auto** (por defecto) o en cualquiera de las unidades de CSS, como píxeles o porcentaje.

column-span—Esta propiedad se aplica a elementos dentro de la caja para determinar si el elemento se ubicará en una columna o repartido entre varias columnas. Los valores disponibles son **all** (todas las columnas) y **none** (por defecto).

column-fill—Esta propiedad determina cómo se repartirá el contenido entre las columnas. Los valores disponibles son **auto** (las columnas son completadas de forma secuencial) y **balance** (el contenido se divide en partes iguales entre todas las columnas).

column-gap—Esta propiedad define el tamaño del espacio entre las columnas. Acepta un valor en cualquiera de las unidades disponibles en CSS, como píxeles y porcentaje.

columns—Esta propiedad nos permite declarar los valores de las propiedades **column-count** y **column-width** al mismo tiempo.

El elemento, cuyo contenido queremos dividir en columnas, es un elemento común, y su contenido se declara del mismo modo que lo hemos hecho antes. El navegador se encarga de

crear las columnas y dividir el contenido por nosotros. El siguiente ejemplo incluye un elemento `<article>` con un texto extenso que vamos a presentar en dos columnas.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Este texto es el título del documento</title>
  <meta charset="utf-8">
  <meta name="description" content="Este es un documento HTML5">
  <meta name="keywords" content="HTML, CSS, JavaScript">
  <link rel="stylesheet" href="misestilos.css">
</head>
<body>
  <section>
    <article id="articulonoticias">
      <span>HTML, sigla en inglés de HyperText Markup Language
(lenguaje de marcas de hipertexto), hace referencia al lenguaje de
marcado para la elaboración de páginas web. Es un estándar que sirve de
referencia del software que conecta con la elaboración de páginas web
en sus diferentes versiones, define una estructura básica y un código
(denominado código HTML) para la definición de contenido de una página
web, como texto, imágenes, vídeos, juegos, entre otros. Es un estándar
a cargo del World Wide Web Consortium (W3C) o Consorcio WWW,
organización dedicada a la estandarización de casi todas las
tecnologías ligadas a la web, sobre todo en lo referente a su escritura
e interpretación.</span>
    </article>
  </section>
</body>
</html>
```

Listado 4-17: Dividiendo artículos en columnas

Las propiedades para generar las columnas se deben aplicar al contenedor. La siguiente regla incluye la propiedad `column-count` para dividir el contenido del elemento `<article>` en dos columnas.

```
#articulonoticias {
  width: 580px;
  padding: 10px;
  border: 1px solid;

  column-count: 2;
  column-gap: 20px;
}
```

Listado 4-18: Definiendo las columnas

La regla del Listado 4-18 asigna un tamaño de 580 píxeles al elemento `<article>` e identifica el área ocupada por su contenedor con un borde sólido. La regla también incluye un relleno de 10 píxeles para separar el texto del borde. El resto de las propiedades dividen el contenido en dos columnas con un espacio intermedio de 20 píxeles. El resultado se muestra en la Figura 4-17

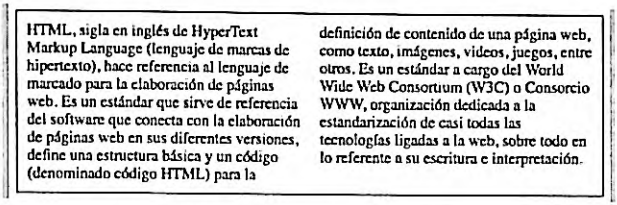

Figura 4-17: Contenido en dos columnas

La propiedad `column-gap` define el tamaño del espacio entre las columnas. Esta separación es solo espacio vacío, pero CSS ofrece las siguientes propiedades para generar una línea que ayuda al usuario a visualizar la división.

column-rule-style—Esta propiedad define el estilo de la línea usada para representar la división. Los valores disponibles son `hidden` (por defecto), `dotted`, `dashed`, `solid`, `double`, `groove`, `ridge`, `inset`, y `outset`.

column-rule-color—Esta propiedad especifica el color de la línea usada para representar la división.

column-rule-width—Esta propiedad especifica el ancho de la línea usada para representar la división.

column-rule—Esta propiedad nos permite definir todos los valores de la línea al mismo tiempo.

Para dibujar una línea en medio de las columnas, tenemos que especificar al menos su estilo. El siguiente ejemplo implementa la propiedad `column-rule` para crear una línea negra de 1 píxel.

```
#articulonoticias {
  width: 580px;
  padding: 10px;
  border: 1px solid;

  column-count: 2;
  column-gap: 20px;
  column-rule: 1px solid #000000;
}
```

Listado 4-19: Agregando una línea entre las columnas

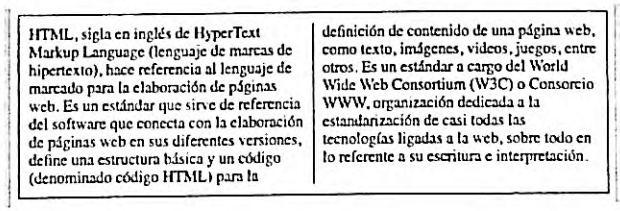

Figura 4-18: Columnas separadas por una línea

Hágalo usted mismo: cree un nuevo archivo HTML con el código del Listado 4-17 y un archivo CSS llamado misestilos.css con el código del Listado 4-18. Abra el documento en su navegador. Debería ver el texto dividido en dos columnas. Reemplace la regla `#articulosnoticias` por la regla del Listado 4-19 y actualice la página en su navegador. Debería ver una línea dividiendo las columnas, tal como muestra la Figura 4-18.

Aplicación de la vida real

El propósito del modelo de caja tradicional es el de organizar la estructura visual de una página web, pero debido a sus características y limitaciones, se deben modificar los documentos para trabajar con este modelo.

Como hemos explicado en el Capítulo 2, los sitios web siguen un patrón estándar y los elementos HTML se han diseñado para crear este diseño, pero no pueden predecir todos los escenarios posibles. Para lograr que los elementos representen las áreas visuales de una página web tradicional, debemos combinarlos con otros elementos. Un truco muy común es envolver los elementos en un elemento contenedor para poder moverlos a las posiciones que deseamos. Por ejemplo, si queremos que la cabecera de nuestro documento sea tan ancha como la ventana del navegador, pero que su contenido se encuentre centrado en la pantalla, podemos envolver el contenido en un elemento `<div>` y luego centrar este elemento en la página. Las siguientes son las adaptaciones que tenemos que hacer al documento introducido en el Capítulo 2 para poder reproducir este tipo de diseño.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Este texto es el título del documento</title>
  <meta charset="utf-8">
  <meta name="description" content="Este es un documento HTML5">
  <meta name="keywords" content="HTML, CSS, JavaScript">
  <link rel="stylesheet" href="misestilos.css">
</head>
<body>
  <header id="cabeceralogo">
    <div>
      <h1>Este es el título</h1>
    </div>
  </header>
  <nav id="menuprincipal">
    <div>
      <ul>
        <li><a href="index.html">Principal</a></li>
        <li><a href="fotos.html">Fotos</a></li>
        <li><a href="videos.html">Videos</a></li>
        <li><a href="contacto.html">Contacto</a></li>
      </ul>
    </div>
  </nav>
  <main>
    <div>
      <section id="articulosprincipales">
```

```
      <article>
        <h1>Título Primer Artículo</h1>
        <time datetime="2016-12-23" pubdate>
          <div class="numerodia">23</div>
          <div class="nombredia">Viernes</div>
        </time>
        <p>Este es el texto de mi primer artículo</p>
        <figure>
          <img src="miimagen.jpg">
        </figure>
      </article>
      <article>
        <h1>Título Segundo Artículo</h1>
        <time datetime="2016-12-7" pubdate>
          <div class="numerodia">7</div>
          <div class="nombredia">Miércoles</div>
        </time>
        <p>Este es el texto de mi segundo artículo</p>
        <figure>
          <img src="miimagen.jpg">
        </figure>
      </article>
    </section>
    <aside id="infoadicional">
      <h1>Información Personal</h1>
      <p>Cita del artículo uno</p>
      <p>Cita del artículo dos</p>
    </aside>
    <div class="recuperar"></div>
  </div>
 </main>
 <footer id="pielogo">
  <div>
    <section class="seccionpie">
      <h1>Sitio Web</h1>
      <p><a href="index.html">Principal</a></p>
      <p><a href="fotos.html">Fotos</a></p>
      <p><a href="videos.html">Videos</a></p>
    </section>
    <section class="seccionpie">
      <h1>Ayuda</h1>
      <p><a href="contacto.html">Contacto</a></p>
    </section>
    <section class="seccionpie">
      <address>Toronto, Canada</address>
      <small>&copy; Derechos Reservados 2016</small>
    </section>
    <div class="recuperar"></div>
  </div>
 </footer>
</body>
</html>
```

Listado 4-20: Definiendo un documento para implementar el modelo de caja tradicional

En este ejemplo, hemos envuelto el contenido de algunos de los elementos estructurales con un elemento `<div>` adicional. Ahora podemos asignar tamaños y alineamientos independientes para cada sección del documento.

Hágalo usted mismo: cree un nuevo archivo HTML con el documento del Listado 4-20 y un archivo CSS llamado misestilos.css para incluir todos los códigos CSS que vamos a presentar a continuación. Abra el documento en su navegador para ver cómo se organizan los elementos por defecto.

Con el documento HTML listo, es hora de desarrollar nuestra hoja de estilo CSS. A este respecto, lo primero que debemos considerar es qué vamos a hacer con los estilos asignados por defecto por el navegador. En la mayoría de los casos, estos estilos no solo son diferentes de lo que necesitamos, sino que además pueden afectar de forma negativa a nuestro diseño. Por ejemplo, los navegadores asignan márgenes a los elementos que usamos frecuentemente en nuestro documento, como el elemento `<p>`. El elemento `<body>` también genera un margen alrededor de su contenido, lo que hace imposible extender otros elementos hasta los límites de la ventana del navegador. Como si esto fuera poco, la forma en la que se configuran los elementos por defecto difiere de un navegador a otro, especialmente cuando consideramos ediciones de navegadores antiguas que aún se encuentran en uso. Para poder crear un diseño coherente, cualquiera que sea el dispositivo en el que se abre, tenemos que resetear algunos de los estilos por defecto, o todos. Una forma práctica de hacerlo es usando un selector CSS llamado *selector universal*. Se trata de un selector que se representa con el carácter * y que referencia todos los elementos del documento. Por ejemplo, la siguiente regla declara un margen y un relleno de 0 píxeles para todos los elementos de nuestro documento.

```
* {
  margin: 0px;
  padding: 0px;
}
```

***Listado 4-21:** Implementando una regla de reseteado*

La primera regla de nuestro archivo CSS introducida en el Listado 4-21 se asegura de que todo elemento tenga un margen y un relleno de 0 píxeles. De ahora en adelante, solo tendremos que modificar los márgenes de los elementos que queremos que sean mayor que cero.

Hágalo usted mismo: para crear la hoja de estilo para el documento del Listado 4-20, tiene que incluir todas las reglas presentadas desde el Listado 4-21, una sobre otra en el mismo archivo (misestilos.css).

Lo básico: lo que hemos hecho con la regla del Listado 4-21 es resetear los estilos de nuestro documento. Esta es una práctica común y generalmente requiere algo más que modificar los márgenes y rellenos, como hemos hecho en este ejemplo. Debido a que los estilos que debemos modificar son en la mayoría de los casos los mismos para cada proyecto, los desarrolladores han creado hojas de estilo que ya incluyen estas reglas y que podemos implementar en nuestros documentos junto con nuestros propios estilos. Estas hojas de estilo se denominan *hojas de estilo de reseteo (Reset*

Style Sheets) y hay varias disponibles. Para ver un ejemplo, visite meyerweb.com/eric/tools/css/reset/.

Con esta simple regla logramos que todos los elementos queden alineados a la izquierda de la ventana del navegador, sin márgenes alrededor y separados por la misma distancia entre ellos.

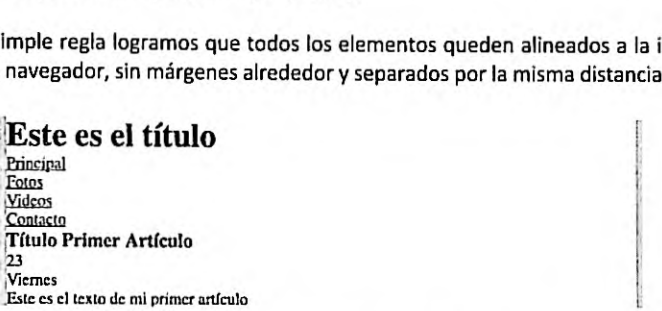

Figura 4-19: Documento con estilos universales

El siguiente paso es diseñar la cabecera. En este caso queremos que el elemento `<header>` se extienda hasta los límites de la ventana del navegador y su contenido esté centrado y se ubique dentro de un área no superior a 960 píxeles (este es un tamaño estándar para las pantallas anchas que tienen los ordenadores de escritorio). Las siguientes son las reglas que se requieren para este propósito.

```
#cabeceralogo {
  width: 96%;
  height: 150px;
  padding: 0% 2%;
  background-color: #0F76A0;
}
#cabeceralogo > div {
  width: 960px;
  margin: 0px auto;
  padding-top: 45px;
}
#cabeceralogo h1 {
  font: bold 54px Arial, sans-serif;
  color: #FFFFFF;
}
```

Listado 4-22: Asignando estilos a la cabecera

Como queremos que la cabecera tenga la anchura de la ventana, tenemos que declarar su tamaño en porcentaje. Cuando el tamaño de un elemento se declara en porcentaje, el navegador se encarga de calcular el tamaño real en píxeles a partir del tamaño actual de su contenedor (en este caso, la ventana del navegador). Para nuestro documento, queremos que la cabecera tenga el mismo ancho que la ventana, pero que incluya un relleno a los lados de modo que su contenido esté separado del borde. Con este propósito, asignamos un valor de 96 % a la propiedad `width` y declaramos un relleno de 2 % a los lados. Si la ventana tiene un ancho de 1000 píxeles, por ejemplo, el elemento `<header>` tendrá un ancho de 960 píxeles y un relleno de 20 píxeles a los lados.

Lo básico: si quiere asignar un margen o un relleno fijo a un elemento pero al mismo tiempo ajustar su ancho para que abarque todo el espacio disponible, puede asignar el valor `auto` a la propiedad `width`. Este valor le pide al navegador que calcule el ancho del elemento a partir del ancho de su contenedor, pero considerando los valores de los márgenes, los rellenos, y los bordes del elemento. Por ejemplo, si la ventana tiene un ancho de 1000 píxeles y asignamos a la cabecera un relleno de 50 píxeles a cada lado y el valor `auto` para su ancho, el navegador le otorgará un ancho de 900 píxeles.

La segunda regla en este ejemplo afecta a los elementos `<div>` que son descendientes directos del elemento `<header>`. Como solo tenemos un único elemento `<div>` que usamos para envolver el contenido de la cabecera, este es el elemento que modificará la regla. Las propiedades asignan un ancho de 960 píxeles y un margen con un valor de 0 píxeles para la parte superior e inferior, y el valor `auto` para el lado izquierdo y derecho. El valor `auto` le pide al navegador que calcule el margen de acuerdo con el tamaño del elemento y el espacio disponible en su contenedor. Esto hace que el navegador centre el elemento `<div>` y, por lo tanto, su contenido, cuando el ancho del contenedor es superior a 960 píxeles.

Finalmente, declaramos la regla `#cabeceralogo h1` para cambiar el tipo de letra y el color del elemento `<h1>` dentro de la cabecera. El resultado se muestra en la Figura 4-20.

Figura 4-20: Cabecera

Lo básico: el límite de 960 píxeles es un valor estándar que se usa para declarar el tamaño de páginas web para las pantallas anchas de los ordenadores personales y portátiles. El valor fue establecido considerando la capacidad de las personas para leer textos extensos. Para que un texto sea legible, se recomienda que tenga un máximo de 50-75 caracteres por línea. Si extendemos todo el contenido hasta los lados de la ventana del navegador, nuestro sitio web no se podría leer en pantallas anchas. Con la introducción de los dispositivos móviles, las limitaciones y requerimientos han cambiado. Los sitios web ya no se desarrollan con diseños fijos. Estudiaremos cómo crear diseños flexibles en la próxima sección de este capítulo y cómo adaptar nuestros sitios web a los dispositivos móviles usando diseño web adaptable en el Capítulo 5.

Al igual que la cabecera, el menú de nuestro diseño se extiende hasta los límites de la ventana, pero las opciones tienen que centrarse en un espacio no superior a 960 píxeles.

```
#menuprincipal {
  width: 96%;
```

```
  height: 50px;
  padding: 0% 2%;
  background-color: #9FC8D9;
  border-top: 1px solid #094660;
  border-bottom: 1px solid #094660;
}
#menuprincipal > div {
  width: 960px;
  margin: 0px auto;
}
```

***Listado 4-23:** Asignando estilos al área de navegación*

Estos estilos posicionan el elemento **<nav>** y su contenido, pero los elementos **<ul>** y **<li>** que conforman el menú todavía tienen asignados los estilos por defecto que crean una lista vertical de ítems, y lo que necesitamos es colocar las opciones una al lado de la otra. Existen varias formas de organizar el elemento horizontalmente, pero la mejor alternativa, en este caso, es declarar los elementos **<li>** como elementos **inline-block**. De este modo, podemos posicionarlos en la misma línea y asignarles un tamaño personalizado.

```
#menuprincipal li {
  display: inline-block;
  height: 35px;
  padding: 15px 10px 0px 10px;
  margin-right: 5px;
}
#menuprincipal li:hover {
  background-color: #6FACC6;
}
#menuprincipal a {
  font: bold 18px Arial, sans-serif;
  color: #333333;
  text-decoration: none;
}
```

***Listado 4-24:** Asignando estilos a las opciones del menú*

La primera regla del Listado 4-24 declara los elementos **<li>** dentro del elemento **<nav>** como elementos **inline-block** y les da una altura de 35 píxeles, con un relleno superior de 15 píxeles y 10 píxeles a los lados. Este ejemplo también incluye una regla con la seudoclase **:hover** para cambiar el color de fondo del elemento **<li>** cada vez que el ratón se encuentra sobre una opción. En la última regla, los elementos **<a>** también se modifican con un color y tipo de letra diferente. El resultado se muestra en la Figura 4-21.

***Figura 4-21:** Menú*

Lo básico: una lista de ítems creada con los elementos `<ul>` y `<li>` tiene asignado por defecto el valor `list-item` para la propiedad `display`. Este modo crea una lista vertical de ítems con gráficos del lado izquierdo que los identifica. Cuando declara un valor diferente para la propiedad `display`, estos gráficos se eliminan. Para modificar o eliminar los indicadores cuando el modo es `list-item`, CSS ofrece la propiedad `list-style` (esta es una propiedad general que define los valores de las propiedades `list-style-image`, `list-style-position`, y `list-style-type`). CSS incluye múltiples valores que podemos asignar a esta propiedad para determinar el tipo de gráfico a mostrar. Los más usados son `none`, `square`, `circle`, y `decimal`. La propiedad también nos permite declarar la posición del gráfico (`inside` o `outside`) e incluso una imagen personalizada (por ejemplo, `list-style: url("migrafico.jpg");`).

A continuación, tenemos que diseñar la sección principal de nuestra página web. Esta sección se ha identificado con el elemento `<main>` y contiene los elementos `<section>` y `<aside>` que necesitamos convertir en columnas. Para seguir el mismo patrón de diseño usado para la cabecera y el menú, tenemos que extender el elemento `<main>` hasta los lados de la ventana y centrar su contenido.

```
main {
  width: 96%;
  padding: 2%;
  background-image: url("fondo.png");
}
main > div {
  width: 960px;
  margin: 0px auto;
}
```

Listado 4-25: Asignando estilos al contenido principal

Las reglas del Listado 4-25 asignan una imagen de fondo al elemento `<main>` para diferenciar el área principal del resto de la página. También agregamos un relleno del 2 % en la parte superior e inferior, de modo que el contenido del área principal se separa del menú y el pie de página.

Lo básico: los fondos cubren el área ocupada por el elemento y su relleno. Por defecto, se considera que los márgenes se encuentran fuera del elemento y, por lo tanto, no se ven afectados por la propiedad `background`. Si quiere que el fondo se muestre en toda el área ocupada por el elemento, tiene que evitar usar márgenes y en su lugar asignar rellenos, como hemos hecho en el Listado 4-25.

Con el área principal ya definida, es hora de crear las dos columnas que presentan el contenido principal de nuestra página web.

```
#articulosprincipales {
  float: left;
  width: 620px;
```

```
  padding-top: 30px;
  background-color: #FFFFFF;
  border-radius: 10px;
}
#infoadicional {
  float: right;
  width: 280px;
  padding: 20px;
  background-color: #E7F1F5;
  border-radius: 10px;
}
#infoadicional h1 {
  font: bold 18px Arial, sans-serif;
  color: #333333;
  margin-bottom: 15px;
}
.recuperar {
  clear: both;
}
```

Listado 4-26: Creando las columnas para el contenido principal

Es este caso, usamos la propiedad `float` para mover hacia los lados los elementos que representan cada columna. El elemento `<section>` se ha movido hacia la izquierda y el elemento `<aside>` hacia la derecha, dejando un espacio entre medio de 20 píxeles.

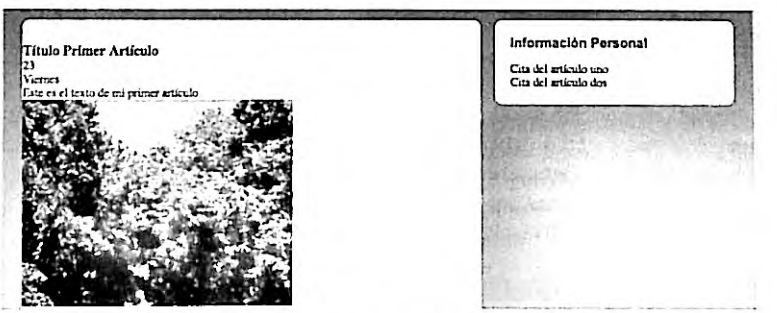

Figura 4-22: Contenido principal

Lo básico: cada vez que los elementos se posicionan con la propiedad `float` debemos acordarnos de recuperar el flujo normal del documento con el elemento siguiente. Con este propósito, en nuestro ejemplo agregamos un elemento `<div>` sin contenido debajo del elemento `<aside>`.

Ahora que las columnas están listas, tenemos que diseñar sus contenidos. El código del Listado 4-26 ya incluye una regla que configura el contenido del elemento `<aside>`, pero aún tenemos que configurar los elementos `<article>` en la primera columna.

Cada elemento `<article>` incluye un elemento `<time>` que representa la fecha en la que el artículo se ha publicado. Para nuestro diseño, decidimos mostrar esta fecha en una caja del lado izquierdo del artículo, por lo que este elemento debe tener una posición absoluta.

```
article {
  position: relative;
  padding: 0px 40px 20px 40px;
}
article time {
  display: block;
  position: absolute;
  top: -5px;
  left: -70px;
  width: 80px;
  padding: 15px 5px;

  background-color: #094660;
  box-shadow: 3px 3px 5px rgba(100, 100, 100, 0.7);
  border-radius: 5px;
}
.numerodia {
  font: bold 36px Verdana, sans-serif;
  color: #FFFFFF;
  text-align: center;
}
.nombredia {
  font: 12px Verdana, sans-serif;
  color: #FFFFFF;
  text-align: center;
}
```

Listado 4-27: Configurando la posición y los estilos del elemento `<time>`

Para asignar una posición absoluta al elemento **<time>**, tenemos que declarar la posición de su elemento padre como relativa. Esto lo logramos asignando el valor **relative** a la propiedad **position** en la primera regla del Listado 4-27. La segunda regla define la posición y el tamaño del elemento **<time>**, el cual se ubicará a 5 píxeles de la parte superior del elemento **<article>** y a 110 píxeles de su lado izquierdo.

Figura 4-23: Posición absoluta para el elemento `<time>`

El resto de las reglas tiene que asignar los estilos requeridos por los elementos restantes dentro de cada elemento **<article>**.

```
article h1 {
  margin-bottom: 5px;
  font: bold 30px Georgia, sans-serif;
}
article p {
  font: 18px Georgia, sans-serif;
}
figure {
  margin: 10px 0px;
}
figure img {
  padding: 5px;
  border: 1px solid;
}
```

Listado 4-28: Asignando estilos a los artículos

Figura 4-24: Artículos

Finalmente, tenemos que agregar unas pocas reglas a nuestra hoja de estilo para configurar el pie de página. El elemento `<footer>`, así como hemos hecho con el resto de los elementos estructurales, se debe extender hasta los lados de la ventana, pero en este caso el contenido se divide con tres elementos `<section>` para presentar la información en columnas. Aunque podríamos crear estas columnas con la propiedad `columns`, debido a que los elementos representan secciones de nuestro documento, la propiedad `float` es más apropiada.

```
#pielogo {
  width: 96%;
  padding: 2%;
  background-color: #0F76A0;
}
#pielogo > div {
  width: 960px;
  margin: 0px auto;
  background-color: #9FC8D9;
  border-radius: 10px;
}
.seccionpie {
  float: left;
  width: 270px;
  padding: 25px;
}
```

```
.seccionpie h1 {
  font: bold 20px Arial, sans-serif;
}
.seccionpie p {
  margin-top: 5px;
}
.seccionpie a {
  font: bold 16px Arial, sans-serif;
  color: #666666;
  text-decoration: none;
}
```

Listado 4-29: Asignando estilos al pie de página

Las reglas del Listado 4-29 definen tres secciones de 276 píxeles cada una, todas flotando hacia la izquierda para formar las tres columnas necesarias para nuestro diseño.

Figura 4-25: Pie de página

En estos casos, donde todas las secciones son iguales, en lugar de declarar los tamaños en píxeles es mejor hacerlo en porcentajes. El valor en porcentaje indica cuánto espacio va a ocupar el elemento dentro del contenedor. Por ejemplo, podemos declarar el ancho de las secciones en el pie de página con un valor de 27.33 % cada una y completar el 100 % con el relleno.

```
.seccionpie {
  float: left;
  width: 27.33%;
  padding: 3%;
}
```

Listado 4-30: Calculando el tamaño de las secciones con valores en porcentajes

Cuando los valores se declaran en porcentajes, el navegador se encarga de calcular cuántos píxeles tienen que asignarse a cada elemento de acuerdo con el espacio disponible en el contenedor. Si aplicamos la regla del Listado 4-30 a nuestro documento, el pie de página se mostrará parecido al de la Figura 4-25, pero los tamaños de las secciones los calculará el navegador antes de presentar la página (960 x 27.33 / 100 = 262).

IMPORTANTE: los valores en porcentaje también pueden ser utilizados para crear diseños flexibles, como veremos en el Capítulo 5, pero existe un modelo mejor para este fin llamado modelo de caja flexible. Estudiaremos el modelo de caja flexible en la siguiente sección de este capítulo.

Con estas reglas hemos finalizado el diseño del documento, pero esta es solo una de las páginas de nuestro sitio web. Este documento representa la página inicial, generalmente

almacenada en el archivo por defecto, como index.html, pero todavía tenemos que crear el resto de los documentos para representar cada página disponible. Afortunadamente, esta no es una tarea complicada. El mismo diseño que hemos desarrollado para la página inicial generalmente se comparte en todo el sitio web, solo tenemos que cambiar el área del contenido principal, pero el resto, como la cabecera, el pie de página y la estructura del documento en general son iguales, incluida la hoja de estilo, que normalmente comparten la mayoría de los documentos.

Hágalo usted mismo: si aún no lo ha hecho, asigne el nombre index.html al documento del Listado 4-20. Cree los archivos fotos.html, videos.html y contacto.html con una copia de este documento. Reemplace el contenido de los elementos `<section>` y `<aside>` en el área principal con el contenido correspondiente a cada página. Si es necesario, agregue nuevas reglas al archivo misestilos.css para ajustar el diseño de cada página. Aunque se recomienda concentrar todos los estilos en un solo archivo, también puede crear diferentes hojas de estilo por cada documento si lo considera adecuado. Abra el archivo index.html en su navegador y haga clic en los enlaces para navegar a través de las páginas.

Si necesitamos definir otras áreas en una página o dividir el área principal en secciones más pequeñas, podemos organizar las cajas con la propiedad `float`, como lo hemos hecho con las secciones dentro del pie de página. Aunque también podemos diseñar algunas áreas con posicionamiento absoluto o relativo, estos modos se reservan para posicionar contenido no relevante, como hemos hecho con las fechas de los artículos en nuestro ejemplo o para mostrar contenido oculto después de recibir una solicitud por parte del usuario. Por ejemplo, podemos crear un menú desplegable que muestra un submenú para cada opción. El siguiente ejemplo ilustra cómo agregar un submenú a la opción **Fotos** de nuestro documento.

```
<ul id="listamenu">
  <li><a href="index.html">Principal</a></li>
  <li><a href="fotos.html">Fotos</a>
    <ul>
      <li><a href="familia.html">Familia</a></li>
      <li><a href="vacaciones.html">Vacaciones</a></li>
    </ul>
  </li>
  <li><a href="videos.html">Videos</a></li>
  <li><a href="contacto.html">Contacto</a></li>
</ul>
```


***Listado 4-31:** Agregando submenús*

Los estilos para un menú que incluye submenus difieren un poco del ejemplo anterior. Tenemos que listar las opciones del submenú de forma vertical en lugar de horizontal, posicionar la lista debajo de la barra del menú con valores absolutos, y solo mostrarla cuando el usuario mueve el ratón sobre la opción principal. Para este propósito, en el código del Listado 4-31, hemos agregado el identificador `listamenu` al elemento `<ul>` que representa el menú principal. Ahora, podemos diferenciar este elemento de los elementos `<ul>` encargados de representar los submenús (solo uno en nuestro ejemplo). Las siguientes reglas usan este identificador para asignar estilos a todos los menús y sus opciones.

```
#listamenu > li {
  position: relative;
  display: inline-block;
  height: 35px;
  padding: 15px 10px 0px 10px;
  margin-right: 5px;
}
#listamenu li > ul {
  display: none;
  position: absolute;
  top: 50px;
  left: 0px;

  background-color: #9FC8D9;
  box-shadow: 3px 3px 5px rgba(100, 100, 100, 0.7);
  border-radius: 0px 0px 5px 5px;

  list-style: none;
  z-index: 1000;
}
#listamenu li > ul > li {
  width: 120px;
  height: 35px;
  padding-top: 15px;
  padding-left: 10px;
}
#listamenu li:hover ul {
  display: block;
}
#listamenu a {
  font: bold 18px Arial, sans-serif;
  color: #333333;
  text-decoration: none;
}
#menuprincipal li:hover {
  background-color: #6FACC6;
}
```

Listado 4-32: Asignando estilos a los submenús

Las primeras dos reglas configuran las opciones principales y sus submenús. En estas reglas, asignamos una posición relativa a los elementos **<li>** que representan las opciones principales y luego especificamos una posición absoluta para los submenús de 50 píxeles debajo de la parte superior del menú. Esto posiciona los submenús debajo de la barra del menú. Para volver los submenús invisibles, declaramos la propiedad **display** con el valor **none**. También hemos tenido que incluir otras propiedades en esta regla como **list-style**, para eliminar los gráficos mostrados por defecto al lado izquierdo de las opciones, y la propiedad **z-index** para asegurarnos de que el submenú siempre se muestra sobre el resto de los elementos del área.

Como hemos hecho anteriormente, para responder al ratón tenemos que implementar la seudoclase **:hover**, pero este caso es algo diferente. Tenemos que mostrar el elemento **<ul>** que representa el submenú cada vez que el usuario mueve el ratón sobre cualquiera de las opciones principales, pero las opciones principales se representan con elementos **<li>**. La solución es

declarar la seudoclase `:hover` para los elementos `<li>`, pero agregar el nombre `ul` al final del selector para asignar los estilos a los elementos `<ul>` que se encuentran dentro del elemento `<li>` afectado por la seudoclase (`#listamenu li:hover ul`). Cuando el usuario mueve el ratón sobre el elemento `<li>` que representa una opción del menú principal, el valor `block` se asigna a la propiedad `display` del elemento `<ul>` y el submenú se muestra en la pantalla.

Figura 4-26: Submenús

Hágalo usted mismo: actualice su documento con el código del Listado 4-31 y las reglas de su hoja de estilo con el código del Listado 4-32. Abra el archivo index.html en su navegador y mueva el ratón sobre la opción **Fotos**. Debería ver algo similar a la Figura 4-26.

4.3 Modelo de caja flexible

El objetivo principal de un modelo de caja es el de ofrecer un mecanismo con el que dividir el espacio disponible en la ventana, y crear las filas y columnas que son parte del diseño de una página web. Sin embargo, las herramientas que ofrece el modelo de caja tradicional no cumplen este objetivo. Definir cómo distribuir las cajas y especificar sus tamaños, por ejemplo, no se puede lograr de forma eficiente con este modelo.

La dificultad en la implementación de patrones de diseño comunes, como expandir columnas para ocupar el espacio disponible, centrar el contenido de una caja en el eje vertical o extender una columna desde la parte superior a la parte inferior de una página, independiente del tamaño de su contenido, han forzado a la industria a buscar otras alternativas. Se han desarrollado varios modelos de caja, pero ninguno ha recibido más atención que el modelo de caja flexible.

IMPORTANTE: aunque el modelo de caja flexible tiene sus ventajas sobre el modelo de caja tradicional, se encuentra aún en estado experimental y algunos navegadores no pueden procesar sus propiedades. Esta es la razón por la que en este capítulo también introducimos el modelo de caja tradicional. Antes de elegir el modelo a aplicar en sus sitios web debe considerar estas limitaciones. Para determinar si puede usar cualquiera de las herramientas que introduce HTML5, visite www.caniuse.com.

Contenedor flexible

El modelo de caja flexible resuelve los problemas del modelo de caja tradicional de una manera elegante. Este modelo aprovecha las herramientas que usa el modelo de caja tradicional, como el posicionamiento absoluto y las columnas, pero en lugar de hacer flotar los

elementos organiza las cajas usando contenedores flexibles. Un contenedor flexible es un elemento que convierte su contenido en cajas flexibles. En este nuevo modelo, cada grupo de cajas debe estar incluido dentro de otra caja que es la encargada de configurar sus características, tal como muestra el siguiente ejemplo.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Este texto es el título del documento</title>
  <meta charset="utf-8">
  <meta name="description" content="Este es un documento HTML5">
  <meta name="keywords" content="HTML, CSS, JavaScript">
  <link rel="stylesheet" href="misestilos.css">
</head>
<body>
  <section id="cajapadre">
    <div id="caja-1">Caja 1</div>
    <div id="caja-2">Caja 2</div>
    <div id="caja-3">Caja 3</div>
    <div id="caja-4">Caja 4</div>
  </section>
</body>
</html>
```

Listado 4-33: Organizando cajas con un contenedor flexible

El documento del Listado 4-33, al igual que los ejemplos anteriores, incluye un elemento **<section>** que actúa como contenedor de otros elementos. La diferencia se presenta en cómo se configuran los elementos desde CSS. Para volver flexibles las cajas dentro del elemento **<section>** (sus tamaños cambian de acuerdo al espacio disponible), tenemos que convertir a este elemento en un contendor flexible. Para este propósito, CSS ofrece los valores **flex** e **inline-flex** para la propiedad **display**. El valor **flex** define un elemento Block flexible y el valor **inline-flex** define un elemento Inline flexible.

```
#cajapadre {
  display: flex;
}
```

Listado 4-34: Convirtiendo el elemento `cajapadre` *en un contenedor flexible*

Hágalo usted mismo: cree un nuevo archivo HTML con el código del Listado 4-33 y un archivo CSS llamado misestilos.css con la regla del Listado 4-34. Abra el documento en su navegador. Debería ver las cajas dentro del elemento **<section>** una al lado de la otra en la misma línea.

Elementos flexibles

Para que un elemento dentro de un contenedor flexible se vuelva flexible, tenemos que declararlo como tal. Las siguientes son las propiedades disponibles para configurar elementos flexibles.

flex-grow—Esta propiedad declara la proporción en la cual el elemento se va a expandir o encoger. La proporción se determina considerando los valores asignados al resto de los elementos de la caja (los elementos hermanos).

flex-shrink—Esta propiedad declara la proporción en la que el elemento se va a reducir. La proporción se determina a partir de los valores asignados al resto de los elementos de la caja (los elementos hermanos).

flex-basis—Esta propiedad declara un tamaño inicial para el elemento.

flex—Esta propiedad nos permite configurar los valores de las propiedades `flex-grow`, `flex-shrink`, y `flex-basis` al mismo tiempo.

Las cajas flexibles se expanden o encogen para ocupar el espacio libre dentro de la caja padre. La distribución del espacio depende de las propiedades del resto de las cajas. Si todas las cajas se configuran como flexibles, el tamaño de cada uno de ellas dependerá del tamaño de la caja padre y del valor de la propiedad `flex`.

```
#cajapadre {
  display: flex;
  width: 600px;
}
#cajapadre > div {
  height: 145px;
  margin: 5px;
  background-color: #CCCCCC;
}
#caja-1 {
  flex: 1;
}
#caja-2 {
  flex: 1;
}
#caja-3 {
  flex: 1;
}
#caja-4 {
  flex: 1;
}
```

Listado 4-35: Convirtiendo las cajas en cajas flexibles con la propiedad `flex`

En el ejemplo del Listado 4-35, solo declaramos el valor `flex-grow` para la propiedad `flex` con el fin de determinar cómo se expandirán las cajas. El tamaño de cada caja se calcula multiplicando el valor del tamaño de la caja padre por el valor de su propiedad `flex` dividido por la suma de los valores `flex-grow` de todas las cajas. Por ejemplo, la fórmula para el elemento `caja-1` es **600 × 1 / 4 = 150**. El valor **600** es el tamaño de la caja padre, **1** es el valor de la propiedad `flex` asignado al elemento `caja-1`, y **4** es la suma de los valores de la propiedad `flex` asignados a cada una de las cajas. Debido a que todas las cajas de nuestro ejemplo tienen el mismo valor **1** en su propiedad `flex`, el tamaño de cada caja será de 150 píxeles menos los márgenes (hemos asignado un margen de 5 píxeles al elemento `<div>`).

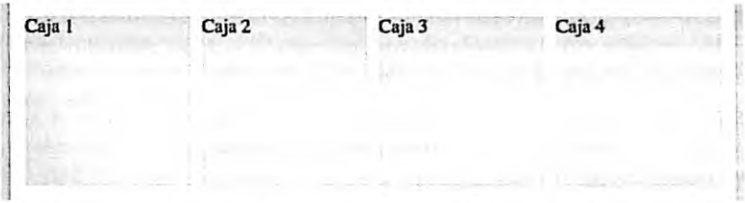

Figura 4-27: Los mismos valores para la propiedad `flex` *distribuyen el espacio equitativamente*

Hágalo usted mismo: reemplace las reglas en su archivo CSS por el código del Listado 4-35 y abra el documento del Listado 4-33 en su navegador. Debería ver algo parecido a la Figura 4-27.

El potencial de esta propiedad es evidente cuando asignamos diferentes valores a cada elemento.

```
#cajapadre {
  display: flex;
  width: 600px;
}
#cajapadre > div {
  height: 145px;
  margin: 5px;
  background-color: #CCCCCC;
}
#caja-1 {
  flex: 2;
}
#caja-2 {
  flex: 1;
}
#caja-3 {
  flex: 1;
}
#caja-4 {
  flex: 1;
}
```

Listado 4-36: Creando una distribución desigual

En el Listado 4-36, asignamos el valor **2** a la propiedad **`flex`** del elemento **`caja-1`**. Ahora, la fórmula para calcular el tamaño de esta caja es **600 × 2 / 5 = 240**. Debido a que no cambiamos el tamaño de la caja padre, el primer valor de la fórmula es el mismo, pero el segundo valor es **2** (el nuevo valor de la propiedad **`flex`** de **`caja-1`**), y la suma de los valores de todas las propiedades **`flex`** es **5** (**2** para **`caja-1`** y **1** para cada una de las otras tres cajas). Aplicando la misma fórmula para el resto de las cajas, podemos obtener sus tamaños: **600 × 1 / 5 = 120**.

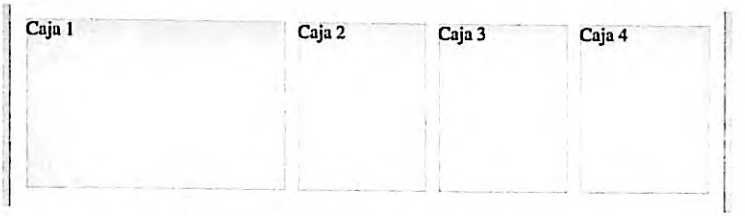

Figura 4-28: Distribución desigual con `flex`

Comparando los resultados, podemos ver cómo se distribuye el espacio. El espacio disponible se divide en porciones de acuerdo a la suma de los valores de la propiedad **`flex`** de cada caja (un total de **5** en nuestro ejemplo). Luego, las porciones se distribuyen entre las cajas. El elemento **`caja-1`** recibe dos porciones porque el valor de su propiedad **`flex`** es **2** y el resto de elementos recibe solo una porción porque el valor de sus propiedades **`flex`** es **1**. La ventaja no es solo que los elementos se vuelven flexibles, sino también que cada vez que agregamos un nuevo elemento, no tenemos que calcular su tamaño; los tamaños de todas las cajas se recalculan automáticamente.

Estas características son interesantes, pero existen otros escenarios posibles. Por ejemplo, cuando una de las cajas es inflexible y tiene un tamaño explícito, las otras cajas se flexionan para compartir el resto del espacio disponible.

```
#cajapadre {
  display: flex;
  width: 600px;
}
#cajapadre > div {
  height: 145px;
  margin: 5px;
  background-color: #CCCCCC;
}
#caja-1 {
  width: 300px;
}
#caja-2 {
  flex: 1;
}

#caja-3 {
  flex: 1;
}
#caja-4 {
  flex: 1;
}
```

Listado 4-37: Combinando cajas flexibles con cajas inflexibles

La primera caja del ejemplo del Listado 4-37 tiene un tamaño de 300 píxeles, por lo que el espacio a distribuir entre el resto de las cajas es de 300 píxeles (**600 – 300 = 300**). El navegador calcula el tamaño de cada caja flexible con la misma fórmula que hemos usado anteriormente: **300 × 1 / 3 = 100**.

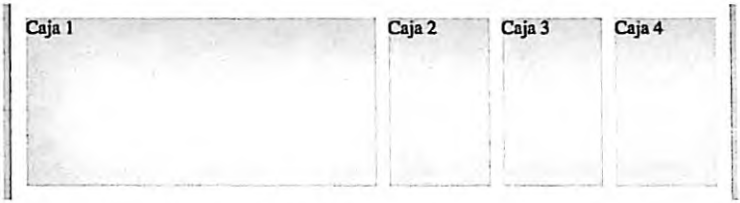

Figura 4-29: Solo se distribuye el espacio libre

Del mismo modo que podemos tener una caja con un tamaño explícito, podemos tener dos o más. El principio es el mismo, solo que el espacio remanente se distribuye entre las cajas flexibles.

Existe un posible escenario en el cual podríamos tener que declarar el tamaño de un elemento pero mantenerlo flexible. Para lograr esta configuración, debemos utilizar el resto de los parámetros disponibles para la propiedad **flex** (**flex-shrink** y **flex-basis**).

```
#cajapadre {
  display: flex;
}
#cajapadre > div {
  height: 145px;
  margin: 5px;
  background-color: #CCCCCC;
}
#caja-1 {
  flex: 1 1 200px;
}
#caja-2 {
  flex: 1 5 100px;
}
#caja-3 {
  flex: 1 5 100px;
}
#caja-4 {
  flex: 1 5 100px;
}
```

Listado 4-38: Controlando cómo se encoge el elemento

Esta vez, se han declarado tres parámetros para la propiedad **flex** de cada caja. El primer parámetro de todas las cajas (**flex-grow**) se ha definido con el valor **1**, declarando la misma proporción de expansión. La diferencia se determina por el agregado de los valores para los parámetros **flex-shrink** y **flex-basis**. El parámetro **flex-shrink** trabaja como **flex-grow**, pero determina la proporción en la que las cajas se reducirán para caber en el espacio disponible. En nuestro ejemplo, el valor de este parámetro es **1** para el elemento **caja-1** y **5** para el resto de las cajas, lo cual asignará más espacio a **caja-1**. El parámetro **flex-basis**, por otro lado, establece un valor inicial para el elemento. Por lo tanto, el valor del parámetro **flex-basis** se considera para calcular cuánto se expandirá o reducirá un elemento flexible. Cuando este valor es 0 o no se declara, el valor considerado es el tamaño del contenido del elemento.

Hágalo usted mismo: reemplace las reglas de su archivo CSS por el código del Listado 4-38. En este ejemplo, no declaramos el tamaño del elemento `cajapadre`, de modo que cuando la ventana del navegador se expande, el elemento padre también se expande y todas las cajas de su interior aumentan en la misma proporción. Por el contrario, cuando el tamaño de la ventana se reduce, el elemento `caja-1` se reduce en una proporción diferente debido al valor especificado para el parámetro `flex-shrink` (1 en lugar de 5).

IMPORTANTE: el valor 0 asignado a los parámetros `flex-grow` o `flex-shrink` no permite que el elemento se expanda o reduzca, respectivamente. Para declarar flexibilidad, los valores de estos parámetros deben ser iguales o mayores que 1.

También podemos asignar el valor `auto` al parámetro `flex-basis` para pedirle al navegador que use el valor de la propiedad `width` como referencia.

```
#cajapadre {
  display: flex;
  width: 600px;
}
#cajapadre > div {
  height: 145px;
  margin: 5px;
  background-color: #CCCCCC;
}
#caja-1 {
  width: 200px;
  flex: 1 1 auto;
}
#caja-2 {
  width: 100px;
  flex: 1 1 auto;
}
#caja-3 {
  width: 100px;
  flex: 1 1 auto;
}
#caja-4 {
  width: 100px;
  flex: 1 1 auto;
}
```

***Listado 4-39:** Definiendo cajas flexibles con un tamaño preferido*

En el Listado 4-39, cada caja tiene un ancho preferido (`width`), pero después de que todas las cajas quedan posicionadas hay un espacio libre de 100 píxeles. Este espacio extra se dividirá entre las cajas flexibles. Para calcular la porción de espacio asignado a cada caja, usamos la misma fórmula que antes: **100 × 1 / 4 = 25**. Esto significa que se agregan 25 píxeles adicionales al tamaño preferido de cada caja (menos los márgenes).

Figura 4-30: Espacio libre distribuido entre las cajas

La propiedad `flex-basis` nos permite definir un tamaño inicial para influenciar al navegador a la hora de distribuir el espacio disponible entre las cajas, pero las cajas aún se expanden o reducen más allá de ese valor. CSS ofrece las siguientes propiedades para poner limitaciones al tamaño de una caja.

max-width—Esta propiedad especifica el ancho máximo permitido para el elemento. Acepta valores en cualquiera de las unidades disponibles en CSS, como píxeles o porcentajes.

min-width—Esta propiedad especifica el ancho mínimo permitido para el elemento. Acepta valores en cualquiera de las unidades disponibles en CSS, como píxeles o porcentajes.

max-height—Esta propiedad especifica la altura máxima permitida para el elemento. Acepta valores en cualquiera de las unidades disponibles en CSS, como píxeles o porcentajes.

min-height—Esta propiedad especifica la altura mínima permitida para el elemento. Acepta valores en cualquiera de las unidades disponibles en CSS, como píxeles o porcentajes.

El siguiente ejemplo declara todas las cajas como flexibles, pero asigna un ancho máximo de 50 píxeles al elemento `caja-1`. Independientemente del espacio disponible, `caja-1` nunca supera los 50 píxeles.

```
#cajapadre {
  display: flex;
}
#cajapadre > div {
  height: 145px;
  margin: 5px;
  background-color: #CCCCCC;
}
#caja-1 {
  max-width: 50px;
  flex: 1;
}
#caja-2 {
  flex: 1;
}
#caja-3 {
  flex: 1;
}
#caja-4 {
  flex: 1;
}
```

Listado 4-40: Declarando un tamaño máximo

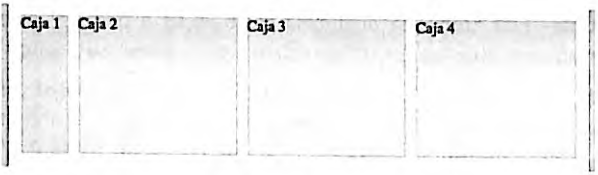

Figura 4-31: Cajas con un tamaño máximo

Organizando elementos flexibles

Por defecto, los elementos dentro de un contenedor flexible se muestran horizontalmente en la misma línea, pero no se organizan con una orientación estándar. Un contenedor flexible usa ejes para describir la orientación de su contenido. La especificación declara dos ejes que son independientes de la orientación: el eje principal y el eje transversal. El eje principal es aquel en el que se presenta el contenido (normalmente es equivalente a la orientación horizontal), y el eje transversal es el perpendicular al eje principal (normalmente equivalente a la orientación vertical). Si la orientación cambia, los ejes se desplazan junto con el contenido.

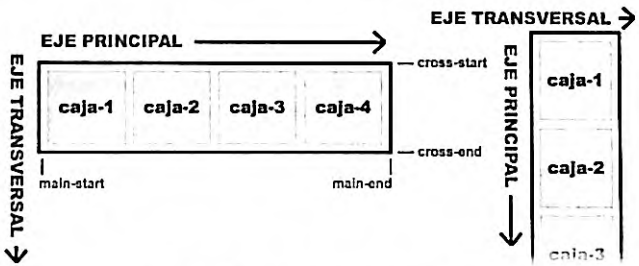

Figura 4-32: Ejes de contenedores flexibles

Las propiedades definidas para este modelo trabajan con estos ejes y organizan los elementos desde sus extremos: main-start, main-end, cross-start y cross-end. La relación entre estos extremos es similar a la relación entre los extremos izquierdo y derecho, o superior e inferior usados para describir la dirección horizontal y vertical en modelos convencionales, pero en este modelo esa relación se invierte cuando la orientación cambia. Cuando uno de estos extremos, como main-start, se menciona en la descripción de una propiedad, debemos recordar que puede referirse al extremo izquierdo o superior, dependiendo de la orientación actual del contenedor (en el diagrama izquierdo de la Figura 4-32, por ejemplo, el extremo main-start está referenciando el lado izquierdo del contenedor, mientras que en el diagrama de la derecha referencia el extremo superior).

Una vez que entendamos cómo trabaja con este modelo, podemos cambiar la organización de las cajas. CSS ofrece las siguientes propiedades con este propósito.

flex-direction—Esta propiedad define el orden y la orientación de las cajas en un contenedor flexible. Los valores disponibles son `row`, `row-reverse`, `column` y `column-reverse`, con el valor `row` configurado por defecto.

order—Esta propiedad especifica el orden de las cajas. Acepta números enteros que determinan la ubicación de cada caja.

justify-content—Esta propiedad determina cómo se va a distribuir el espacio libre. Los valores disponibles son **flex-start**, **flex-end**, **center**, **space-between**, y **space-around**.

align-items—Esta propiedad alinea las cajas en el eje transversal. Los valores disponibles son **flex-start**, **flex-end**, **center**, **baseline**, y **stretch**.

align-self—Esta propiedad alinea una caja en el eje transversal. Trabaja como **align-items** pero afecta cajas de forma individual. Los valores disponibles son **auto**, **flex-start**, **flex-end**, **center**, **baseline**, y **stretch**.

flex-wrap—Esta propiedad determina si se permiten crear múltiples líneas de cajas. Los valores disponibles son **nowrap**, **wrap**, y **wrap-reverse**.

align-content—Esta propiedad alinea las líneas de cajas en el eje vertical. Los valores disponibles son **flex-start**, **flex-end**, **center**, **space-between**, **space-around**, y **stretch**.

Si lo que necesitamos es configurar la dirección de las cajas, podemos usar la propiedad **flex-direction**. Esta propiedad se asigna al contenedor con un valor que corresponde al orden que queremos otorgar al contenido. El valor **row** declara la orientación de las cajas de acuerdo a la orientación del texto (normalmente horizontal) y ordena las cajas desde main-start a main-end (normalmente de izquierda a derecha). El valor **row-reverse** declara la misma orientación que **row**, pero invierte el orden de los elementos desde main-end a main-start (normalmente de derecha a izquierda). El valor **column** declara la orientación de acuerdo a la orientación en la cual se presentan los bloques de texto (normalmente vertical) y ordena las cajas desde main-start a main-end (normalmente de extremo superior a inferior). Y finalmente, el valor **column-reverse** declara la misma orientación que **column**, pero invierte el orden de los elementos desde main-end a main-start (normalmente de extremo inferior a superior). El siguiente ejemplo revierte el orden natural de una línea de cajas.

```
#cajapadre {
  display: flex;
  flex-direction: row-reverse;
}
#cajapadre > div {
  height: 145px;
  margin: 5px;
  background-color: #CCCCCC;
}
#caja-1 {
  flex: 1;
}
#caja-2 {
  flex: 1;
}
#caja-3 {
  flex: 1;
}
#caja-4 {
  flex: 1;
}
```

Listado 4-41: *Invirtiendo la orientación de las cajas*

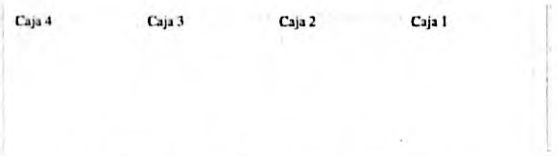

Figura 4-33: Cajas en orden invertido

Lo básico: CSS ofrece una propiedad llamada `writing-mode` que determina la orientación de las líneas de texto (horizontal o vertical), y esta es la razón por la cual el resultado de las propiedades del modelo de caja flexible siempre depende de la orientación establecida previamente para el texto. Para obtener más información acerca de esta propiedad, visite nuestro sitio web y siga los enlaces de este capítulo.

El orden de las cajas también se puede personalizar. La propiedad `order` nos permite declarar la ubicación de cada caja.

```
#cajapadre {
  display: flex;
}
#cajapadre > div {
  height: 145px;
  margin: 5px;
  background-color: #CCCCCC;
}
#caja-1 {
  flex: 1;
  order: 2;
}
#caja-2 {
  flex: 1;
  order: 4;
}
#caja-3 {
  flex: 1;
  order: 3;
}
#caja-4 {
  flex: 1;
  order: 1;
}
```

***Listado 4-42:** Definiendo la posición de cada caja*

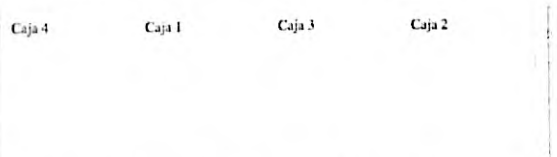

Figura 4-34: Nuevas posiciones para cada caja definidas por la propiedad `order`

Una característica importante del modelo de caja flexible es la capacidad de distribuir el espacio libre. Cuando las cajas no ocupan todo el espacio en el contenedor, dejan espacio libre que debe ubicarse en alguna parte del diseño. Por ejemplo, las siguientes reglas declaran un tamaño de 600 píxeles para el contenedor flexible y un ancho de 100 píxeles para cada caja, dejando un espacio libre de 200 píxeles (menos los márgenes).

```
#cajapadre {
  display: flex;
  width: 600px;
  border: 1px solid;
}
#cajapadre > div {
  height: 145px;
  margin: 5px;
  background-color: #CCCCCC;
}
#caja-1 {
  width: 100px;
}
#caja-2 {
  width: 100px;
}
#caja-3 {
  width: 100px;
}
#caja-4 {
  width: 100px;
}
```

***Listado 4-43:** Distribuyendo el espacio libre en un contenedor flexible*

El ejemplo del Listado 4-43 agrega un borde al contenedor para poder identificar el espacio extra. Por defecto, las cajas se ordenan desde main-start a main-end (normalmente de izquierda a derecha), dejando un espacio libre al final.

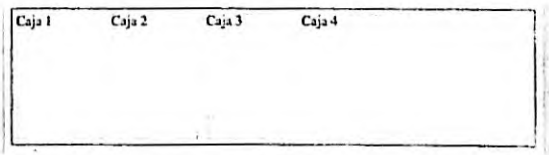

***Figura 4-35:** Cajas y espacio libre dentro de un contendor flexible*

Este comportamiento se puede modificar con la propiedad `justify-content`. El valor por defecto asignado a esta propiedad es `flex-start`, que ordena las cajas según se muestra en la Figura 4-35, aunque podemos asignar un valor diferente para personalizar la forma en la que se distribuyen las cajas y el espacio libre. Por ejemplo, el valor `flex-end` desplaza el espacio al comienzo del contenedor y las cajas hacia el final.

```
#cajapadre {
  display: flex;
```

```
  width: 600px;
  border: 1px solid;
  justify-content: flex-end;
}
#cajapadre > div {
  height: 145px;
  margin: 5px;
  background-color: #CCCCCC;
}
#caja-1 {
  width: 100px;
}
#caja-2 {
  width: 100px;
}
#caja-3 {
  width: 100px;
}
#caja-4 {
  width: 100px;
}
```

Listado 4-44: *Distribuyendo el espacio vacío con la propiedad* `justify-content`

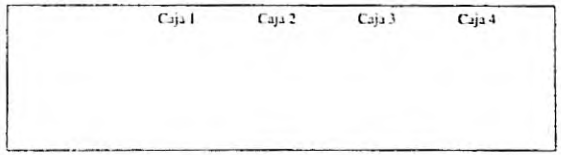

Figura 4-36: *Espacio vacío distribuido con* `justify-content: flex-end`

Las siguientes figuras muestran el efecto que produce el resto de los valores disponibles para esta propiedad.

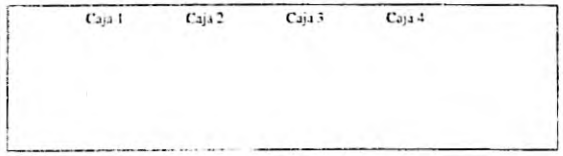

Figura 4-37: *Espacio vacío distribuido con* `justify-content: center`

Figura 4-38: *Espacio vacío distribuido con* `justify-content: space-between`

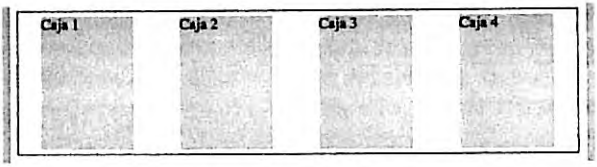

Figura 4-39: Espacio vacío distribuido con `justify-content: space-around`

Otra propiedad que puede ayudarnos a distribuir espacio es **`align-items`**. Esta propiedad trabaja como **`justify-content`** pero alinea las cajas en el eje transversal. Esta característica logra que la propiedad sea aproplada para realizar una alineación vertical.

```
#cajapadre {
  display: flex;
  width: 600px;
  height: 200px;
  border: 1px solid;
  align-items: center;
}
#cajapadre > div {
  height: 145px;
  margin: 5px;
  background-color: #CCCCCC;
}
#caja-1 {
  flex: 1;
}
#caja-2 {
  flex: 1;
}
#caja-3 {
  flex: 1;
}
#caja-4 {
  flex: 1;
}
```

Listado 4-45: Distribuyendo el espacio vertical

En el Listado 4-45 hemos definido la altura del contenedor, dejando un espacio libre de 55 píxeles. Debido a que asignamos el valor **`center`** a la propiedad **`align-items`**, este espacio se distribuye hacia el extremo superior e inferior, tal como muestra la Figura 4-40.

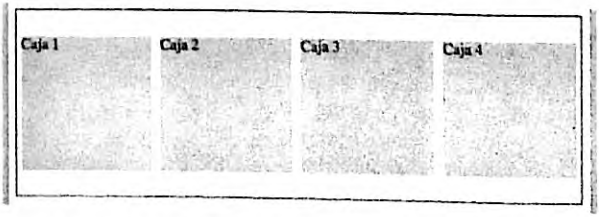

Figura 4-40: Alineación vertical con `align-items: center`

Los valores disponibles para la propiedad **align-items** son **flex-start**, **flex-end**, **center**, **baseline** y **stretch**. El último valor estira las cajas desde el extremo superior al inferior para adaptarlas al espacio disponible. Esta característica es tan importante que el valor **stretch** se declara por defecto para todos los contenedores flexibles. El efecto que logra el valor **stretch** es que, cada vez que no se declara la altura de las cajas, estas adoptan automáticamente el tamaño de sus elementos padre.

```
#cajapadre {
  display: flex;
  width: 600px;
  height: 200px;
  border: 1px solid;
  align-items: stretch;
}
#cajapadre > div {
  margin: 5px;
  background-color: #CCCCCC;
}
#caja-1 {
  flex: 1;
}
#caja-2 {
  flex: 1;
}
#caja-3 {
  flex: 1;
}
#caja-4 {
  flex: 1;
}
```

Listado 4-46: Estirando las cajas para ocupar el espacio vertical disponible

Figura 4-41: Estirando las cajas con `align-items: stretch`

Esta característica es extremadamente útil cuando nuestro diseño presenta columnas con diferente contenido. Si dejamos que el contenido determine la altura, una columna será más corta que la otra. Asignando el valor **stretch** a la propiedad **align-items**, las columnas más cortas se estiran para coincidir con las más largas.

Esta propiedad también ofrece el valor **flex-start** para alinear las cajas al comienzo de la línea, el cual queda determinado por la orientación del contenedor (normalmente el extremo izquierdo o superior).

```
#cajapadre {
  display: flex;
  width: 600px;
  height: 200px;
  border: 1px solid;
  align-items: flex-start;
}
#cajapadre > div {
  height: 145px;
  margin: 5px;
  background-color: #CCCCCC;
}
#caja-1 {
  flex: 1;
}
#caja-2 {
  flex: 1;
}
#caja-3 {
  flex: 1;
}
#caja-4 {
  flex: 1;
}
```

***Listado 4-47:** Alineando las cajas hacia el extremo superior*

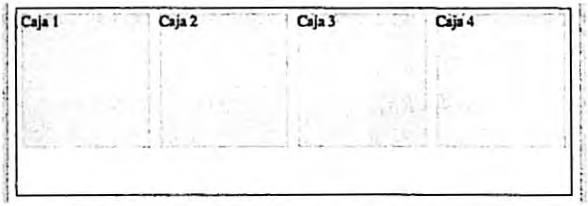

***Figura 4-42:** Cajas alineadas con* `align-items: flex-start`

El valor **flex-end** alinea las cajas hacia el final del contenedor (normalmente el extremo derecho o inferior).

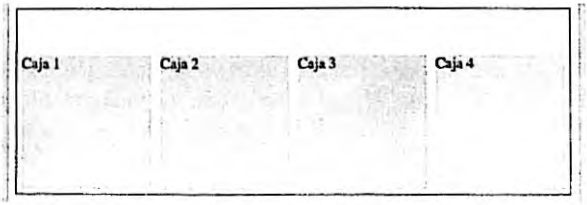

***Figura 4-43:** Cajas alineadas con* `align-items: flex-end`

Finalmente, el valor **baseline** alinea las cajas por la línea base de la primera línea de contenido. El siguiente ejemplo asigna un tipo de letra diferente al contenido del elemento **caja-2** para mostrar el efecto producido por este valor.

```
#cajapadre {
  display: flex;
  width: 600px;
  height: 200px;
  border: 1px solid;
  align-items: baseline;
}
#cajapadre > div {
  height: 145px;
  margin: 5px;
  background-color: #CCCCCC;
}
#caja-1 {
  flex: 1;
}
#caja-2 {
  flex: 1;
  font-size: 36px;
}
#caja-3 {
  flex: 1;
}
#caja-4 {
  flex: 1;
}
```

***Listado 4-48:** Alineando las cajas por la línea base*

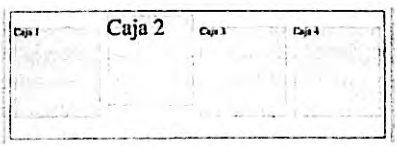

***Figura 4-44:** Cajas alineadas con* `align-items: baseline`

A veces puede resultar útil alinear las cajas de forma independiente, sin importar la alineación establecida por el contenedor. Esto se puede lograr asignando la propiedad **align-self** a la caja que queremos modificar.

```
#cajapadre {
  display: flex;
  width: 600px;
  height: 200px;
  border: 1px solid;
  align-items: flex-end;
}
#cajapadre > div {
  height: 145px;
```

```
  margin: 5px;
  background-color: #CCCCCC;
}
#caja-1 {
  flex: 1;
}
#caja-2 {
  flex: 1;
  align-self: center;
}
#caja-3 {
  flex: 1;
}
#caja-4 {
  flex: 1;
}
```

Listado 4-49: Cambiando la alineación del elemento `caja-2`

Las reglas del Listado 4-49 alinean los elementos hacia el extremo inferior del contenedor, excepto el elemento **caja-2**, que se alinea hacia el centro por la propiedad **align-self**.

Figura 4-45: Caja alineada con `align-self`

Un contenedor flexible puede organizar las cajas en una o varias líneas. La propiedad **flex-wrap** declara esta condición usando tres valores: **nowrap**, **wrap** y **wrap-reverse**. El valor **nowrap** define el contenedor flexible como un contenedor de una sola línea (las líneas no se agrupan), el valor **wrap** define el contenedor como un contenedor de múltiples líneas y las ordena desde el extremo cross-start a cross-end, mientras que el valor **wrap-reverse** genera múltiples líneas en orden invertido.

```
#cajapadre {
  display: flex;
  width: 600px;
  border: 1px solid;
  justify-content: center;
  flex-wrap: wrap;
}
#cajapadre > div {
  height: 40px;
  margin: 5px;
  background-color: #CCCCCC;
}
#caja-1 {
  width: 100px;
}
```

```
#caja-2 {
  width: 100px;
}
#caja-3 {
  width: 100px;
}
#caja-4 {
  flex: 1 1 400px;
}
```

Listado 4-50: Creando dos líneas de cajas con la propiedad `flex-wrap`

En el Listado 4-50, las primeras tres cajas tienen un tamaño de 100 píxeles, suficiente como para ubicarlas en una sola línea dentro de un contenedor de 600 píxeles de ancho, pero la última caja se declara como flexible con un tamaño inicial de 400 píxeles (`flex-basis`) y, por lo tanto, no hay espacio suficiente en el contendor para ubicar todas las cajas en una sola línea. El navegador tiene dos opciones: puede reducir el tamaño de la caja flexible para ubicarla en el espacio disponible o generar una nueva línea. Debido a que la propiedad `flex-wrap` se ha declarado con el valor `wrap`, se crea una nueva línea, como en la Figura 4-46.

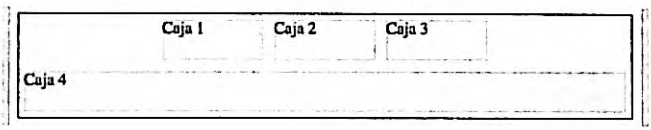

Figura 4-46: Múltiples líneas en un contenedor flexible

El elemento `caja-4` se ha declarado como flexible por la propiedad `flex`, por lo que no solo se ubica en una nueva línea, sino que también se expande hasta ocupar todo el espacio disponible (el valor de 400 píxeles declarado por el parámetro `flex-basis` es solo el ancho sugerido, no una declaración de tamaño). Aprovechando el espacio libre que deja la última caja, las tres primeras cajas quedan alineadas con la propiedad `justify-content`.

El orden de las líneas se puede invertir con el valor `wrap-reverse`, tal como se ilustra a continuación.

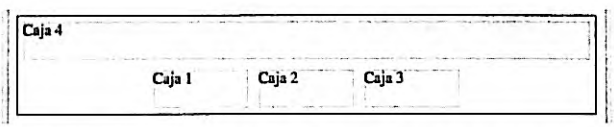

Figura 4-47: Nuevo ordenamiento de líneas usando `flex-wrap: wrap-reverse`

Cuando tenemos contenedores con múltiples líneas, es posible que necesitemos alinearlas. CSS ofrece la propiedad `align-content` para alinear líneas de cajas en un contenedor flexible.

```
#cajapadre {
  display: flex;
  width: 600px;
  height: 200px;
  border: 1px solid;
  flex-wrap: wrap;
```

```
  align-content: flex-start;
}
#cajapadre > div {
  height: 50px;
  margin: 5px;
  background-color: #CCCCCC;
}
#caja-1 {
  flex: 1 1 100px;
}
#caja-2 {
  flex: 1 1 100px;
}
#caja-3 {
  flex: 1 1 100px;
}
#caja-4 {
  flex: 1 1 400px;
}
```

Listado 4-51: Alineando múltiples líneas con la propiedad `align-content`

La propiedad **`align-content`** puede tener seis valores: **`flex-start`**, **`flex-end`**, **`center`**, **`space-between`**, **`space-around`** y **`stretch`**. El valor **`stretch`** se declara por defecto y lo que hace es expandir las líneas para llenar el espacio disponible a menos que se haya declarado un tamaño fijo para los elementos.

En el ejemplo del Listado 4-51, todas las cajas se declaran como flexibles con un ancho y una altura inicial de 50 píxeles, y el elemento **`cajapadre`** se define como un contenedor de múltiples líneas con la propiedad **`flex-wrap`**. Esto crea un contenedor flexible con dos líneas, similar al del ejemplo anterior, pero con espacio vertical suficiente para experimentar.

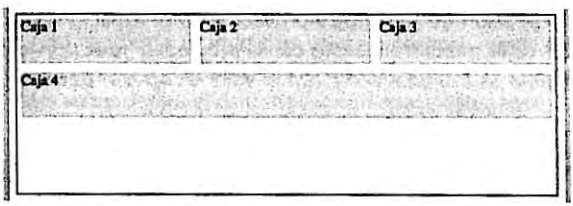

Figura 4-48: Líneas alineadas con `align-content: flex-start`

Las siguientes figuras muestran el efecto producido por el resto de los valores disponibles para la propiedad **`align-content`**.

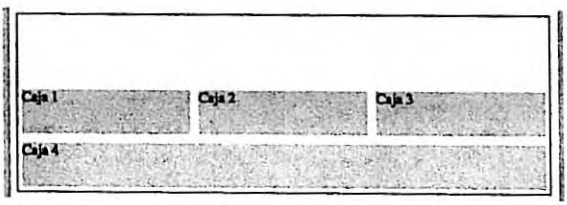

Figura 4-49: Líneas alineadas con `align-content: flex-end`

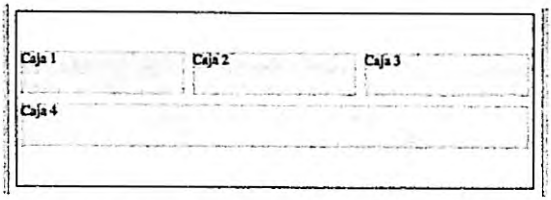

Figura 4-50: Líneas alineadas con `align-content: center`

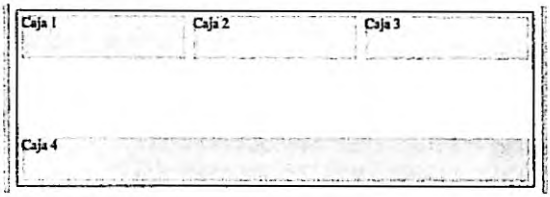

Figura 4-51: Líneas alineadas con `align-content: space-between`

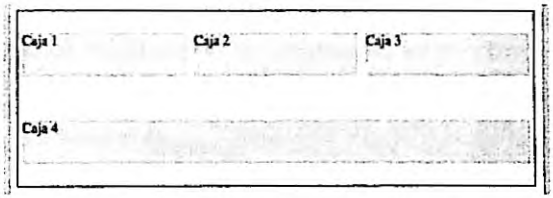

Figura 4-52: Líneas alineadas con `align-content: space-around`

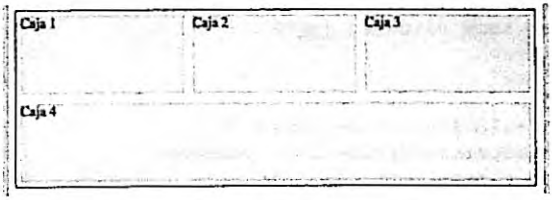

Figura 4-53: Líneas alineadas con `align-content: stretch`

Aplicación de la vida real

No existe mucha diferencia entre un documento diseñado con el modelo de caja tradicional y el que tenemos que usar para implementar el modelo de caja flexible. Por ejemplo, para diseñar una página web con el modelo de caja flexible a partir del documento del Listado 4-20, solo tenemos que eliminar los elementos ** ** que hemos para recuperar el flujo normal del documento. El resto del documento permanece igual.

```
<!DOCTYPE html>
<html lang="es">
<head>
```

```
  <title>Este texto es el título del documento</title>
  <meta charset="utf-8">
  <meta name="description" content="Este es un documento HTML5">
  <meta name="keywords" content="HTML, CSS, JavaScript">
  <link rel="stylesheet" href="misestilos.css">
</head>
<body>
  <header id="cabeceralogo">
    <div>
      <h1>Este es el título</h1>
    </div>
  </header>
  <nav id="menuprincipal">
    <div>
      <ul>
        <li><a href="index.html">Principal</a></li>
        <li><a href="fotos.html">Fotos</a></li>
        <li><a href="videos.html">Videos</a></li>
        <li><a href="contacto.html">Contacto</a></li>
      </ul>
    </div>
  </nav>
  <main>
    <div>
      <section id="articulosprincipales">
        <article>
          <h1>Título Primer Artículo</h1>
          <time datetime="2016-12-23" pubdate>
            <div class="numerodia">23</div>
            <div class="nombredia">Viernes</div>
          </time>
          <p>Este es el texto de mi primer artículo</p>
          <figure>
            <img src="miimagen.jpg">
          </figure>
        </article>
        <article>
          <h1>Título Segundo Artículo</h1>
          <time datetime="2016-12-7" pubdate>
            <div class="numerodia">7</div>
            <div class="nombredia">Miércoles</div>
          </time>
          <p>Este es el texto de mi segundo artículo</p>
          <figure>
            <img src="miimagen.jpg">
          </figure>
        </article>
      </section>
      <aside id="infoadicional">
        <h1>Información Personal</h1>
        <p>Cita del artículo uno</p>
        <p>Cita del artículo dos</p>
      </aside>
    </div>
  </main>
  <footer id="pielogo">
```

```
    <div>
      <section class="seccionpie">
        <h1>Sitio Web</h1>
        <p><a href="index.html">Principal</a></p>
        <p><a href="fotos.html">Fotos</a></p>
        <p><a href="videos.html">Videos</a></p>
      </section>

      <section class="seccionpie">
        <h1>Ayuda</h1>
        <p><a href="contacto.html">Contacto</a></p>
      </section>
      <section class="seccionpie">
        <address>Toronto, Canada</address>
        <small>&copy; Derechos Reservados 2016</small>
      </section>
    </div>
  </footer>
</body>
</html>
```

Listado 4-52: Definiendo un documento para aplicar el modelo de caja flexible

Como la organización de los elementos estructurales es la misma, las reglas CSS que tenemos que aplicar al documento también son similares. Solo tenemos que transformar los elementos estructurales en contenedores flexibles y volver flexible su contenido cuando el diseño lo demanda. Por ejemplo, las siguientes reglas asignan el modo **flex** para la propiedad **display** del elemento **<header>** y declaran al elemento ** ** dentro de este elemento como flexible, para que la cabecera y su contenido se expandan hasta ocupar el espacio disponible en la ventana.

```
* {
  margin: 0px;
  padding: 0px;
}
#cabeceralogo {
  display: flex;
  justify-content: center;
  width: 96%;
  height: 150px;
  padding: 0% 2%;
  background-color: #0F76A0;
}
#cabeceralogo > div {
  flex: 1;
  max-width: 960px;
  padding-top: 45px;
}
#cabeceralogo h1 {
  font: bold 54px Arial, sans-serif;
  color: #FFFFFF;
}
```

Listado 4-53: Convirtiendo la cabecera en un contenedor flexible

Como queremos que el contenido de la cabecera quede centrado en la pantalla, tenemos que asignar el valor **center** a la propiedad **justify-content**. El elemento ** ** se ha declarado flexible, lo que significa que se expandirá o reducirá de acuerdo con el espacio disponible, pero, como hemos explicado antes, cuando la página se presenta en dispositivos con pantalla ancha, tenemos que asegurarnos de que no sea demasiado ancho y al usuario le resulte incómodo leer su contenido. Esto se resuelve con la propiedad **max-width**. Gracias a esta propiedad, el elemento ** ** no se expandirá más de 960 píxeles.

El mismo procedimiento se debe aplicar a nuestro menú, pero el resto de los elementos dentro del elemento **<nav>** usan las mismas propiedades y valores implementados anteriormente.

```
#menuprincipal {
  display: flex;
  justify-content: center;
  width: 96%;
  height: 50px;
  padding: 0% 2%;
  background-color: #9FC8D9;
  border-top: 1px solid #094660;
  border-bottom: 1px solid #094660;
}
#menuprincipal > div {
  flex: 1;
  max-width: 960px;
}
#menuprincipal li {
  display: inline-block;
  height: 35px;
  padding: 15px 10px 0px 10px;
  margin-right: 5px;
}
#menuprincipal li:hover {
  background-color: #6FACC6;
}
#menuprincipal a {
  font: bold 18px Arial, sans-serif;
  color: #333333;
  text-decoration: none;
}
```

Listado 4-54: Convirtiendo el menú en un contenedor flexible

Las reglas para el contenido principal también son las mismas, pero esta vez tenemos dos contenedores flexibles, uno representado por el elemento **<main>** responsable de centrar el contenido en la página, y otro representado por el elemento ** ** encargado de configurar las dos columnas creadas por los elementos **<section>** y **<aside>**. Por esta razón, la regla que afecta al elemento ** ** requiere ambas propiedades: **display** para declarar el elemento como un contenedor flexible y **flex** para declarar el contenedor mismo como un elemento flexible.

```
main {
  display: flex;
  justify-content: center;
```

```
  width: 96%;
  padding: 2%;
  background-image: url("fondo.png");
}
main > div {
  display: flex;
  flex: 1;
  max-width: 960px;
}
#articulosprincipales {
  flex: 1;
  margin-right: 20px;
  padding-top: 30px;
  background-color: #FFFFFF;
  border-radius: 10px;
}
#infoadicional {
  width: 280px;
  padding: 20px;
  background-color: #E7F1F5;
  border-radius: 10px;
}
#infoadicional h1 {
  font: bold 18px Arial, sans-serif;
  color: #333333;
  margin-bottom: 15px;
}
```

Listado 4-55: Convirtiendo las columnas en contenedores flexibles

El elemento **<aside>** se ha declarado con un ancho fijo, lo que significa que solo la columna de la izquierda, representada por el elemento **<section>**, se va a expandir o reducir para ocupar el resto del espacio disponible.

Las propiedades para el contenido del elemento **<section>** (la columna izquierda) son exactamente las mismas que las que hemos definido para el modelo de caja tradicional.

```
article {
  position: relative;
  padding: 0px 40px 20px 40px;
}
article time {
  display: block;
  position: absolute;
  top: -5px;
  left: -70px;
  width: 80px;
  padding: 15px 5px;
  background-color: #094660;
  box-shadow: 3px 3px 5px rgba(100, 100, 100, 0.7);
  border-radius: 5px;
}
.numerodia {
  font: bold 36px Verdana, sans-serif;
  color: #FFFFFF;
```

```
  text-align: center;
}
.nombredia {
  font: 12px Verdana, sans-serif;
  color: #FFFFFF;
  text-align: center;
}
article h1 {
  margin-bottom: 5px;
  font: bold 30px Georgia, sans-serif;
}
article p {
  font: 18px Georgia, sans-serif;
}
figure {
  margin: 10px 0px;
}
figure img {
  max-width: 98%;
  padding: 1%;
  border: 1px solid;
}
```

Listado 4-56: Configurando el contenido

El pie de página presenta un desafío similar al contenido principal. Tenemos que convertir al elemento **<footer>** en una caja flexible y declarar como flexibles las tres columnas creadas en su interior para presentar la información.

```
#pielogo {
  display: flex;
  justify-content: center;
  width: 96%;
  padding: 2%;
  background-color: #0F76A0;
}
#pielogo > div {
  display: flex;
  flex: 1;
  max-width: 960px;
  background-color: #9FC8D9;
  border-radius: 10px;
}
.seccionpie {
  flex: 1;
  padding: 25px;
}
.seccionpie h1 {
  font: bold 20px Arial, sans-serif;
}
.seccionpie p {
  margin-top: 5px;
}
```

```
.seccionpie a {
  font: bold 16px Arial, sans-serif;
  color: #666666;
  text-decoration: none;
}
```

Listado 4-57: Convirtiendo al pie de página en un contenedor flexible

Hágalo usted mismo: cree un nuevo archivo HTML con el código del Listado 4-52 y un archivo CSS llamado misestilos.css con las reglas introducidas desde el Listado 4-53. Recuerde incluir la imagen miimagen.jpg en el mismo directorio. Abra el documento en su navegador. Debería ver la misma página creada anteriormente con el modelo de caja tradicional, con la excepción de que esta vez las secciones de la página son flexibles (sus tamaños cambian a medida que el tamaño de la ventana se incrementa o reduce) y la columna generada por el elemento `<aside>` se extiende hasta el extremo inferior del área principal.

Capítulo 5
Diseño web adaptable

5.1 Web móvil

La introducción en el mercado del iPhone en el año 2007 cambió el mundo para siempre. Hasta ese momento, solo disponíamos de móviles con pantallas pequeñas y sin la capacidad suficiente de descargar y mostrar sitios web. Si queríamos navegar en la Web, teníamos que hacerlo en un ordenador personal. Esto presentaba una situación fácil de controlar para los desarrolladores. Los usuarios contaban con un único tipo de pantalla y esta tenía el tamaño suficiente para incluir todo lo que necesitaban. Las resoluciones más populares del momento incluían al menos 1024 píxeles horizontales, lo cual permitía a los desarrolladores diseñar sus sitios web con un tamaño estándar (como el que presentamos en el capítulo anterior). Pero cuando el iPhone apareció en el mercado, los sitios web con un diseño fijo como estos no se podían leer en una pantalla tan pequeña. La primera solución fue desarrollar dos sitios web separados, uno para ordenadores y otro para iPhones, pero la introducción en el mercado de nuevos dispositivos, como el iPad en el año 2010, forzó nuevamente a los desarrolladores a buscar mejores alternativas. Dar flexibilidad a los elementos fue una de las soluciones implementadas, pero no resultó suficiente. Las páginas web con varias columnas no se mostraban bien en pantallas pequeñas. La solución final debía incluir elementos flexibles, además de la posibilidad de adaptar el diseño a cada dispositivo. Así es como nació lo que hoy conocemos como diseño web adaptable, o por su nombre en inglés *responsive web design*.

El diseño web adaptable es una técnica que combina diseños flexibles con una herramienta provista por CSS llamada *Media Queries* (consulta de medios), que nos permite detectar el tamaño de la pantalla y realizar los cambios necesarios para adaptar el diseño a cada situación.

Media Queries

Una Media Query es una regla reservada en CSS que se incorporó con el propósito de permitir a los desarrolladores detectar el medio en el que se muestra el documento. Por ejemplo, usando Media Queries podemos determinar si el documento se muestra en un monitor o se envía a una impresora, y asignar los estilos apropiados para cada caso. Para este propósito, las Media Queries ofrecen las siguientes palabras clave.

all—Las propiedades se aplican en todos los medios.

print—Las propiedades se aplican cuando la página web se envía a una impresora.

screen—Las propiedades se aplican cuando la página web se muestra en una pantalla color.

speech—Las propiedades se aplican cuando la página web se procesa por un sintetizador de voz.

Lo que hace que las Media Queries sean útiles para el diseño web es que también pueden detectar el tamaño del medio. Con las Media Queries podemos detectar el tamaño del área de visualización (la parte de la ventana del navegador donde se muestran nuestras páginas web) y definir diferentes reglas CSS para cada dispositivo. Existen varias palabras clave que podemos usar para detectar estas características. Las siguientes son las que más se usan para desarrollar un sitio web con diseño web adaptable.

width—Esta palabra clave determina el ancho en que se aplican las propiedades.

height—Esta palabra clave determina la altura a la que se aplican las propiedades.

min-width—Esta palabra clave determina el ancho mínimo desde el cual que se aplican las propiedades.

max-width—Esta palabra clave determina el ancho máximo hasta el cual que se aplican las propiedades.

aspect-ratio—Esta palabra clave determina la proporción en la cual que se aplican las propiedades.

orientation—Esta palabra clave determina la orientación en la cual que se aplican las propiedades. Los valores disponibles son **portrait** (vertical) y **landscape** (horizontal).

resolution—Esta palabra clave determina la densidad de píxeles en la cual que se aplican las propiedades. Acepta valores en puntos por pulgada (dpi), puntos por centímetro (dpcm) o por proporción en píxeles (dppx). Por ejemplo, para detectar una pantalla de tipo retina con una escala de 2, podemos usar el valor **2dppx**.

Usando estas palabras clave, podemos detectar ciertos aspectos del medio y del área de visualización en los que se va a mostrar la página y modificar las propiedades para ajustar el diseño a las condiciones actuales. Por ejemplo, si definimos la Media Query con la palabra clave **width** y el valor **768px**, las propiedades solo se aplicarán cuando la página se muestra en un iPad estándar en modo portrait (orientación vertical).

Para definir una Media Query, podemos declarar solo las palabras clave y los valores que necesitamos. Las palabras clave que describen una característica, como el ancho del área de visualización, tienen que declararse entre paréntesis. Si se incluye más de una palabra clave, podemos asociarlas con los operadores lógicos **and** (y) y **or** (o). Por ejemplo, la Media Query **all and (max-width: 480px)** asigna las propiedades al documento en todos los medios, pero solo cuando el área de visualización tiene un ancho de 480 píxeles o menos.

Existen dos maneras de declarar Media Queries: desde el documento usando el atributo **media** del elemento **<link>**, o desde la hoja de estilo con la regla **@media**. Cuando usamos el elemento **<link>**, podemos seleccionar el archivo CSS con la hoja de estilo que queremos cargar para una configuración específica. Por ejemplo, el siguiente documento carga dos archivos CSS, uno que contiene los estilos generales que aplicaremos en toda situación y medios, y otro con los estilos requeridos para presentar la página en un dispositivo de pantalla pequeña (480 píxeles de ancho o menos).

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Este texto es el título del documento</title>
  <meta charset="utf-8">
  <meta name="description" content="Este es un documento HTML5">
  <meta name="keywords" content="HTML, CSS, JavaScript">
  <link rel="stylesheet" href="adaptabletodos.css">
  <link rel="stylesheet" media="(max-width: 480px)"
href="adaptablecelulares.css">
</head>
<body>
```

```
  <section>
    <div>
      <article>
        <h1>Título Primer Artículo</h1>
        <img src="miimagen.jpg">
      </article>
    </div>
  </section>
  <aside>
    <div>
      <h1>Información</h1>
      <p>Cita del artículo uno</p>
      <p>Cita del artículo dos</p>
    </div>
  </aside>
  <div class="recuperar"></div>
</body>
</html>
```

Listado 5-1: Cargando estilos con Media Queries

Cuando el navegador lee este documento, primero carga el archivo adaptabletodos.css y luego, si el ancho del área de visualización es de 480 píxeles o inferior, carga el archivo adaptablecelulares.css y aplica los estilos al documento.

CSS son las iniciales del nombre hojas de estilo en cascada (*Cascading Style Sheets*), lo cual significa que las propiedades se procesan en cascada y, por lo tanto, las nuevas propiedades reemplazan a las anteriores. Cuando tenemos que modificar un elemento para un área de visualización particular, solo necesitamos declarar las propiedades que queremos cambiar, pero los valores del resto de las propiedades definidas anteriormente permanecen iguales. Por ejemplo, podemos asignar un color de fondo al cuerpo del documento en el archivo adaptabletodos.css y luego cambiarlo en el archivo acaptablecelulares.css, pero este nuevo valor solo se aplicará cuando el documento se presenta en una pantalla pequeña. La siguiente es la regla que implementaremos en el archivo adaptabletodos.css.

```
body {
  margin: 0px;
  padding: 0px;
  background-color: #990000;
}
```

Listado 5-2: Definiendo los estilos por defecto (adaptabletodos.css)

Los estilos definidos en el archivo adaptablecelulares.css tienen que modificar solo los valores de las propiedades que queremos cambiar (en este caso la propiedad **background-color**), pero el resto de los valores debe permanecer igual a como se han declarado en el archivo adaptabletodos.css.

```
body {
  background-color: #3333FF;
}
```

Listado 5-3: Definiendo los estilos para pantallas pequeñas (adaptablecelulares.css)

Hágalo usted mismo: cree un nuevo archivo HTML con el documento del Listado 5-1. Cree un archivo CSS llamado adaptabletodos.css con la regla del Listado 5-2 y un archivo CSS llamado adaptablecelulares.css con la regla del Listado 5-3. Abra el documento en su navegador. Debería ver la página con un fondo rojo. Arrastre un lado de la ventana para reducir su tamaño. Cuando el ancho del área de visualización sea de 480 píxeles o inferior, debe ver el color del fondo cambiar a azul.

Con el elemento `<link>` y el atributo `media` podemos cargar diferentes hojas de estilo para cada situación, pero cuando solo se modifican unas pocas propiedades, podemos incluir todas en la misma hoja de estilo y definir las Media Queries con la regla `@media`, tal como ilustra el siguiente ejemplo.

```
body {
  margin: 0px;
  padding: 0px;
  background-color: #990000;
}
@media (max-width: 480px) {
  body {
    background-color: #3333FF;
  }
}
```

Listado 5-4: Declarando las Media Queries con la regla `@media` *(*adaptabletodos.css*)*

La Media Query se declara con la regla `@media` seguida de las palabras clave y valores que definen las características del medio y las propiedades entre llaves. Cuando el navegador lee esta hoja de estilo, asigna las propiedades de la regla `body` al elemento `<body>` y luego, si el ancho del área de visualización es de 480 píxeles o inferior, cambia el color de fondo del elemento `<body>` a `#3333FF` (azul).

Hágalo usted mismo: reemplace las reglas en su archivo adaptabletodos.css por el código del Listado 5-4 y elimine el segundo elemento `<link>` del documento del Listado 5-1 (en este ejemplo, solo necesita un archivo CSS para todos sus estilos). Abra el documento en su navegador y reduzca el tamaño de la ventana para ver cómo CSS aplica la regla definida por la Media Query cuando el ancho del área de visualización es de 480 píxeles o menos.

Puntos de interrupción

En el ejemplo anterior, declaramos una Media Query que detecta si el ancho del área de visualización es igual o menor a 480 píxeles y aplica las propiedades si se satisface la condición. Esta es una práctica común. Los dispositivos disponibles en el mercado estos días presentan una gran variedad de pantallas de diferentes tamaños. Si usamos la palabra clave `width` para detectar un dispositivo específico, algunos dispositivos que no conocemos o se acaban de introducir en el mercado no se detectarán y mostrarán nuestro sitio web de forma incorrecta. Por ejemplo, la Media Query `width: 768px` detecta solo los iPads de tamaño estándar en modo portrait (vertical) porque solo esos dispositivos en esa orientación especifica presentan

ese tamaño. Un iPad en modo landscape (horizontal), o un iPad Pro, que presenta una resolución diferente, o cualquier otro dispositivo con una pantalla más pequeña o más grande de 768 píxeles, no se verá afectado por estas propiedades, incluso aunque el tamaño varíe solo uno o dos píxeles. Para evitar errores, no debemos seleccionar tamaños específicos. En su lugar, debemos establecer puntos máximos o mínimos en los cuales el diseño debe cambiar significativamente, y utilizar el modelo de caja para adaptar el tamaño de los elementos al espacio disponible cuando el ancho de la pantalla se encuentra entre estos puntos. Estos puntos máximos y mínimos se especifican con las palabras clave **max-width** y **min-width**, y se llaman *puntos de interrupción* (*breakpoints* en inglés).

Los puntos de interrupción son aquellos en los que el diseño de una página web requiere cambios significativos para poder adaptarse al tamaño de la pantalla. Estos puntos son los tamaños en los cuales la flexibilidad de los elementos no es suficiente para ajustar las cajas al espacio disponible y tenemos que mover los elementos de un lugar a otro, o incluso excluirlos del diseño para que nuestro sitio web siga siendo legible. No existen estándares establecidos que podamos seguir para determinar estos puntos; todo depende de nuestro diseño. Por ejemplo, podríamos decidir que cuando el ancho del área de visualización es igual o menor que 480 píxeles tenemos que usar solo una columna para presentar la información, pero esto solo es posible si el diseño original contiene más de una columna. Los valores más comunes son 320, 480, 768, y 1024.

Para establecer un punto de interrupción, tenemos que declarar una Media Query con las palabras clave **max-width** o **min-width**, de modo que las propiedades se aplicarán cuando el tamaño del área de visualización supere estos límites. La palabra clave que aplicamos depende de la dirección que queramos seguir. Si queremos diseñar primero para dispositivos con pantalla pequeña, debemos establecer tamaños mínimos con **min-width**, pero si queremos establecer un diseño genérico aplicable a las pantallas anchas de ordenadores personales y luego modificar las propiedades a medida que el tamaño se reduce, lo mejor es establecer máximos con **max-width**. Por ejemplo, la siguiente hoja de estilo asigna un color de fondo por defecto al cuerpo del documento, pero a medida que el tamaño de la pantalla se reduce, el color se modifica.

```
body {
  margin: 0px;
  padding: 0px;
  background-color: #990000;
}
@media (max-width: 1024px) {
  body {
    background-color: #3333FF;
  }
}
@media (max-width: 768px) {
  body {
    background-color: #FF33FF;
  }
}
@media (max-width: 480px) {
  body {
    background-color: #339933;
  }
}
@media (max-width: 320px) {
  body {
```

```
    background-color: #CCCCCC;
  }
}
```

Listado 5-5: Incluyendo múltiples puntos de interrupción

Cuando se asignan los estilos del Listado 5-5 al documento del Listado 5-1, el navegador presenta la página con un fondo rojo en un área de visualización superior a 1024 píxeles, pero cambia de color cada vez que el ancho se encuentra por debajo de los límites establecidos por las Media Queries.

Hágalo usted mismo: reemplace las reglas del archivo adaptabletodos.css con el código del Listado 5-5. Abra el documento en su navegador y modifique el tamaño de la ventana. Si el área de visualización es superior a 1024 píxeles, el cuerpo se muestra con un fondo rojo, pero si el tamaño se reduce a 1024 píxeles o menos, el color cambia a azul, a 768 píxeles o menos, cambia a rosado, a 480 píxeles o menos cambia a verde, y a 320 píxeles o menos cambia a gris.

Área de visualización

El área de visualización (*viewport* en inglés) es la parte de la ventana del navegador donde se muestran nuestras páginas web. Debido a la relación entre la ventana y el área de visualización, ambos deberían presentar el mismo tamaño en dispositivos móviles, pero esto no se cumple en todos los casos. Por defecto, algunos dispositivos asignan un ancho de 980 píxeles al área de visualización, sin importar su tamaño real o el tamaño real de la pantalla. Esto significa que las Media Queries de nuestras hojas de estilo verán un ancho de 980 píxeles cuando en realidad el tamaño del área de visualización es totalmente diferente. Para normalizar esta situación y forzar al navegador a definir el tamaño del área de visualización igual al tamaño real de la pantalla, tenemos que declarar el elemento `<meta>` en la cabecera de nuestros documentos con el nombre `viewport` y valores que determinan el ancho y la escala que queremos ver. Los dos valores requeridos son `width` e `initial-scale` para declarar el ancho del área de visualización y su escala. El siguiente ejemplo ilustra cómo debemos configurar el elemento `<meta>` para pedirle al navegador que defina el tamaño y la escala reales de la pantalla del dispositivo.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Este texto es el título del documento</title>
  <meta charset="utf-8">
  <meta name="description" content="Este es un documento HTML5">
  <meta name="keywords" content="HTML, CSS, JavaScript">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="adaptabletodos.css">
</head>
<body>
  <section>
    <div>
      <article>
```

```
        <h1>Título Primer Artículo</h1>
        <img src="miimagen.jpg">
      </article>
    </div>
  </section>
  <aside>
    <div>
      <h1>Información</h1>
      <p>Cita del artículo uno</p>
      <p>Cita del artículo dos</p>
    </div>
  </aside>
  <div class="recuperar"></div>
</body>
</html>
```

Listado 5-6: Configurando el área de visualización con el elemento `<meta>`

Lo básico: el elemento `<meta>` que declara el área de visualización puede incluir otros valores para configurar atributos como las escalas mínimas y máximas (`minimum-scale` y `maximum-scale`), o si queremos permitir a los usuarios ampliar o reducir la página web (`user-scalable`). Para más información, visite nuestro sitio web y siga los enlaces de este capítulo.

Flexibilidad

En un diseño web adaptable tenemos que permitir que nuestras página sean flexibles, de modo que los elementos se adapten al espacio disponible cuando el ancho del área de visualización se encuentra entre los puntos de interrupción. Esto se puede lograr con el modelo de caja flexible: solo tenemos que crear contenedores flexibles y declarar sus tamaños como flexibles con la propiedad `flex` (ver Capítulo 4), pero este modelo de caja no lo reconocen algunos navegadores y, por lo tanto, la mayoría de los desarrolladores aún implementan el modelo de caja tradicional. Aunque el modelo de caja tradicional no fue diseñado para trabajar con elementos flexibles, podemos volver flexibles los elementos declarando sus tamaños en porcentaje. Cuando declaramos el tamaño de un elemento en porcentaje, el navegador calcula el tamaño real en píxeles a partir del tamaño de su contenedor. Por ejemplo, si asignamos un ancho de 80 % a un elemento que se encuentra dentro de otro elemento con un tamaño de 1000 píxeles, el ancho del elemento será de 800 píxeles (80 % de 1000). Debido a que el tamaño de los elementos cambia cada vez que lo hace el tamaño de sus contenedores, estos se vuelven flexibles.

El siguiente ejemplo crea dos columnas con los elementos `<section>` y `<aside>` del documento del Listado 5-6, y declara sus anchos en porcentaje para hacerlos flexibles.

```
* {
  margin: 0px;
  padding: 0px;
}
section {
  float: left;
  width: 70%;
  background-color: #999999;
}
```

```
aside {
  float: left;
  width: 30%;
  background-color: #CCCCCC;
}
.recuperar {
  clear: both;
}
```

Listado 5-7: Creando elementos flexibles con valores en porcentaje

Las reglas del Listado 5-7 declaran el tamaño de los elementos **<section>** y **<aside>** al 70 % y al 30 % del tamaño de su contenedor, respetivamente. Si el tamaño de la pantalla cambia, el navegador recalcula el tamaño de los elementos y, por lo tanto, estos elementos se expanden o reducen de acuerdo con el espacio disponible.

Figura 5-1: Elementos flexibles con el modelo de caja tradicional

Cada vez que declaramos el ancho del elemento en porcentajes, tenemos que asegurarnos de que la suma total de los valores no exceda el 100 %. De otro modo, los elementos no entrarán en el espacio disponible y se moverán a una nueva línea. En el ejemplo del Listado 5-7, esto fue fácil de lograr porque solo teníamos dos elementos sin márgenes, rellenos o bordes, pero tan pronto como agregamos un valor adicional, como el relleno, tenemos que recordar restar estos valores al ancho del elemento o el total excederá el 100 %. Por ejemplo, las siguientes reglas reducen el ancho de los elementos **<section>** y **<aside>** para poder agregar un relleno de 2 % a cada lado y un margen de 5 % entre medio.

```
* {
  margin: 0px;
  padding: 0px;
}
section {
  float: left;
  width: 61%;
  padding: 2%;
  margin-right: 5%;
  background-color: #999999;
}
aside {
  float: left;
  width: 26%;
```

```
  padding: 2%;
  background-color: #CCCCCC;
}
.recuperar {
  clear: both;
}
```

Listado 5-8: Agregando márgenes y relleno a elementos flexibles

Las reglas del Listado 5-8 asignan al elemento `<section>` un ancho de 61 %, un relleno de 2 % del lado superior, derecho, inferior e izquierdo, y un margen de 5 % del lado derecho, y al elemento `<aside>` un ancho de 26 % y un relleno de 2 % del lado superior, derecho, inferior e izquierdo. Debido a que la suma de estos valores no excede el 100 % (61 + 2 + 2 + 5 + 26 + 2 + 2 = 100), los elementos se colocan lado a lado en la misma línea.

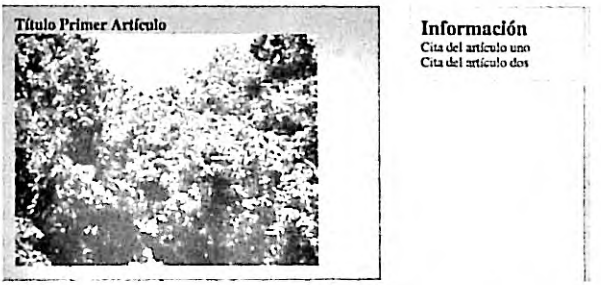

Figura 5-2: Elementos flexibles con relleno y márgenes

Hágalo usted mismo: reemplace las reglas del archivo adaptabletodos.css por el código del Listado 5-8. Incluya la imagen miimagen.jpg en el directorio del documento. Abra el documento en su navegador. Debería ver algo similar a la Figura 5-2.

Box-sizing

Cada vez que se calcula el total del área ocupada por un elemento, el navegador obtiene el valor final con la fórmula tamaño + márgenes + relleno + bordes. Si declaramos la propiedad `width` con un valor de 100 píxeles, `margin` a 20 píxeles, `padding` a 10 píxeles y `border` a 1 píxel, el total del área horizontal ocupada por el elemento será `100 + 40 + 20 + 2 = 162` píxeles (los valores de las propiedades `margin`, `padding` y `border` se han duplicado en la fórmula porque asumimos que se han asignado los mismos valores a los lados izquierdo y derecho de la caja). Esto significa que cada vez que declaramos el tamaño de un elemento con la propiedad `width`, tenemos que recordar que el área que se necesita en la página para mostrar el elemento puede ser mayor. Esta situación es particularmente problemática cuando los tamaños se definen en porcentajes, o cuando intentamos combinar valores en porcentaje con otros tipos de unidades como píxeles. CSS incluye la siguiente propiedad para modificar este comportamiento.

box-sizing—Esta propiedad nos permite decidir cómo se calculará el tamaño de un elemento y forzar a los navegadores a incluir el relleno y el borde en el tamaño declarado por la propiedad `width`. Los valores disponibles son `content-box` y `border-box`.

Por defecto, el valor de la propiedad **box-sizing** se declara como **content-box**, lo cual significa que el navegador agregará los valores de las propiedades **padding** y **border** al tamaño especificado por la propiedad **width**. Si en cambio asignamos a esta propiedad el valor **border-box**, el navegador incluye el relleno y el borde como parte del ancho, y entonces disponemos de otras opciones como combinar anchos definidos en porcentaje con rellenos en píxeles, tal como se muestra en el siguiente ejemplo.

```
* {
  margin: 0px;
  padding: 0px;
}
section {
  float: left;
  width: 65%;
  padding: 20px;
  margin-right: 5%;
  background-color: #999999;
  box-sizing: border-box;
}
aside {
  float: left;
  width: 30%;
  padding: 20px;
  background-color: #CCCCCC;
  box-sizing: border-box;
}
.recuperar {
  clear: both;
}
```

Listado 5-9: Incluyendo el relleno y el borde en el tamaño del elemento

En el Listado 5-9 declaramos la propiedad **box-sizing** con el valor **border-box** para ambos elementos, **<section>** y **<aside>**. Debido a esto, ya no tenemos que considerar el relleno cuando calculamos el tamaño de los elementos (65 + 5 + 30 = 100).

Hágalo usted mismo: reemplace las reglas del archivo adaptabletodos.css con el código del Listado 5-9. Abra el documento en su navegador. Debería ver algo similar a la Figura 5-2, pero esta vez el relleno no cambia cuando se modifica el tamaño de la ventana porque su valor se ha definido en píxeles.

Fijo y flexible

En los anteriores ejemplos hemos visto cómo definir cajas flexibles con rellenos fijos, pero muchas veces nos encontraremos con la necesidad de tener que combinar columnas completas con valores flexibles y fijos. Nuevamente, esto es muy fácil de lograr con el modelo de caja flexible; simplemente tenemos que crear un contenedor flexible, declarar el tamaño de los elementos que queremos que sean flexibles con la propiedad **flex** y aquellos que queremos que tengan un tamaño fijo con la propiedad **width** (ver Capítulo 4, Listado 4-37). Pero el modelo de caja tradicional no se diseñó para realizar esta clase de tareas. Por suerte existen algunos

trucos que podemos implementar para lograr este propósito. La alternativa más popular es la de declarar un relleno en la columna flexible con el que hacemos sitio para ubicar la columna de tamaño fijo y agregar un margen negativo que desplaza esta columna al espacio vacío dejado por el relleno. Las siguientes son las reglas que necesitamos para declarar un ancho flexible para el elemento **<section>** y un ancho fijo para el elemento **<aside>** de nuestro documento.

```
* {
  margin: 0px;
  padding: 0px;
}
section {
  float: left;
  width: 100%;
  padding-right: 260px;
  margin-right: -240px;
  box-sizing: border-box;
}
section > div {
  padding: 20px;
  background-color: #999999;
}
aside {
  float: left;
  width: 240px;
  padding: 20px;
  background-color: #CCCCCC;
  box-sizing: border-box;
}
.recuperar {
  clear: both;
}
```

Listado 5-10: Declarando columnas fijas y flexibles

Las reglas del Listado 5-10 asignan un ancho de 100 % y un relleno del lado derecho de 260 píxeles al elemento **<section>**. Como usamos la propiedad **box-sizing** para incluir el relleno en el valor de la propiedad **width**, el contenido del elemento ocupará solo el lado izquierdo, dejando un espacio vacío a la derecha donde podemos ubicar la columna con tamaño fijo. Finalmente, para mover la columna creada por el elemento **<aside>** a este espacio, declaramos un margen negativo de 240 píxeles en el lado derecho del elemento **<section>**.

Lo básico: en este ejemplo hacemos flotar ambas columnas a la izquierda, lo que significa que se van a ubicar una al lado de la otra en la misma línea. En casos como estos, podemos crear un espacio en medio de las columnas incrementando el valor de la propiedad **padding**. En el ejemplo del Listado 5-10 declaramos un relleno de 260 píxeles y un margen de 240 píxeles, lo que significa que la columna de tamaño fijo solo se moverá 240 píxeles hacia la izquierda, dejando un espacio entre medio de 20 píxeles.

Debido a que estamos usando el relleno para generar un espacio vacío en el que ubicar la columna de la derecha, tenemos que asignar el relleno normal de la columna y el color de

fondo al elemento contenedor `<div>` dentro del elemento `<section>`. Esto no solo nos permite agregar un relleno al contenido de la columna, sino además asignar un color de fondo solo al área ocupada por el contenido, diferenciando las dos columnas en la pantalla.

Figura 5-3: Columnas flexibles y fijas

Hágalo usted mismo: reemplace las reglas en su archivo adaptabletodos.css por el código del Listado 5-10. Abra el documento en su navegador. Arrastre un lado de la ventana para cambiar su tamaño. Debería ver la columna izquierda expandirse o encogerse, y la columna derecha siempre del mismo tamaño (240 píxeles).

Si queremos ubicar la columna de tamaño fijo a la izquierda en lugar de a la derecha, el proceso es el mismo, pero tenemos que declarar la columna izquierda debajo de la columna derecha en el código (como se han declarado en el Listado 5-6) y luego configurar sus posiciones con la propiedad `float`, tal como hacemos en el siguiente ejemplo.

```
* {
  margin: 0px;
  padding: 0px;
}
section {
  float: right;
  width: 100%;
  padding-left: 260px;
  margin-left: -240px;
  box-sizing: border-box;
}
section > div {
  padding: 20px;
  background-color: #999999;
}
aside {
  float: left;
  width: 240px;
  padding: 20px;
  background-color: #CCCCCC;
  box-sizing: border-box;
}
```

```
.recuperar {
  clear: both;
}
```

Listado 5-11: Moviendo la columna fija a la izquierda

El elemento **<section>** se declara primero en nuestro documento, pero debido a que asignamos el valor **right** a su propiedad **float**, se muestra del lado derecho de la pantalla. La posición del elemento en el código asegura que se va a presentar sobre el elemento **<aside>**, y el valor de la propiedad **float** lo mueve al lado que queremos en la página. El resto del código es similar al del ejemplo anterior, excepto que esta vez tenemos que modificar el relleno y el margen del elemento **<section>** del lado izquierdo en lugar del derecho.

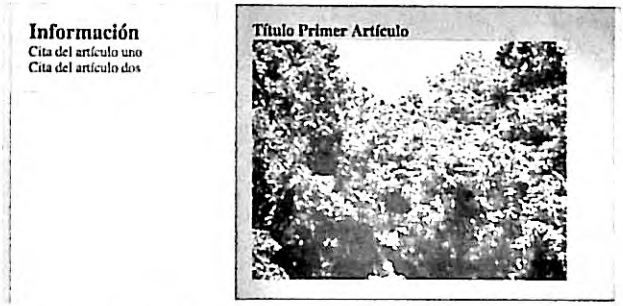

Figura 5-4: Columna fija del lado izquierdo

Lo básico: cuando movemos un elemento a una nueva posición asignando un margen negativo al siguiente elemento, el navegador presenta el primer elemento detrás del segundo y, por lo tanto, el primer elemento no recibe ningún evento, como los clics del usuario en los enlaces. Para asegurarnos de que la columna de tamaño fijo permanece visible y accesible al usuario, tenemos que declararla en el código debajo de la columna flexible. Esta es la razón por la que para mover la columna creada por el elemento **<aside>** a la izquierda no hemos tenido que modificar el documento. La columna se ubica en su lugar dentro de la página por medio de la propiedad **float**.

Si lo que queremos es declarar dos columnas con tamaños fijos a los lados y una columna flexible en el medio usando este mismo mecanismo, tenemos que agrupar las columnas dentro de un contenedor y luego aplicar las propiedades a los elementos dos veces, dentro y fuera del contenedor. En el siguiente documento, agrupamos dos elementos **<section>** dentro de un elemento ** ** identificado con el nombre **contenedor** para contener las columnas izquierda y central. Estas dos columnas se han identificado con los nombres **columnaizquierda** y **columnacentral**.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Este texto es el título del documento</title>
```

```
  <meta charset="utf-8">
  <meta name="description" content="Este es un documento HTML5">
  <meta name="keywords" content="HTML, CSS, JavaScript">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="misestilos.css">
</head>
<body>
  <div id="contenedor">
    <section id="columnacentral">
      <div>
        <article>
          <h1>Título Primer Artículo</h1>
          <img src="miimagen.jpg">
        </article>
      </div>
    </section>
    <section id="columnaizquierda">
      <div>
        <h1>Opciones</h1>
        <p>Opción 1</p>
        <p>Opción 2</p>
      </div>
    </section>
    <div class="recuperar"></div>
  </div>
  <aside>
    <div>
      <h1>Información</h1>
      <p>Cita del artículo uno</p>
      <p>Cita del artículo dos</p>
    </div>
  </aside>
</body>
</html>
```

Listado 5-12: Creando un documento con dos columnas fijas

El código CSS para este documento es sencillo. Los elementos **contenedor** y **columnacentral** tienen que declararse como flexibles, por lo que tenemos que asignar al contenedor el relleno y el margen adecuados para ubicar el elemento **<aside>** a la derecha, y luego definir el relleno y el margen del elemento **columnacentral** para hacer sitio para la columna fija del lado izquierdo.

```
* {
  margin: 0px;
  padding: 0px;
}
#contenedor {
  float: left;
  width: 100%;
  padding-right: 260px;
  margin-right: -240px;
  box-sizing: border-box;
}
```

```
aside {
  float: left;
  width: 240px;
  padding: 20px;
  background-color: #CCCCCC;
  box-sizing: border-box;
}
#columnacentral {
  float: right;
  width: 100%;
  padding-left: 220px;
  margin-left: -200px;
  box-sizing: border-box;
}
#columnacentral > div {
  padding: 20px;
  background-color: #999999;
}
#columnaizquierda {
  float: left;
  width: 200px;
  padding: 20px;
  background-color: #CCCCCC;
  box-sizing: border-box;
}
.recuperar {
  clear: both;
}
```


Listado 5-13: *Definiendo dos columnas fijas a los lados*

En este ejemplo declaramos el elemento **columnaizquierda** después del elemento **columnacentral**. Con esto nos aseguramos de que **columnaizquierda** se presenta sobre **columnacentral** y, por lo tanto, es accesible. Como hemos hecho antes, las posiciones en la página web de estos elementos se definen con la propiedad **float**. En consecuencia, el elemento **columnaizquierda** se coloca en el lado izquierdo de la página con un ancho fijo de 200 píxeles, el elemento **columnacentral** se coloca en el centro con un ancho flexible y el elemento **<aside>** a la derecha con un ancho fijo de 240 píxeles.

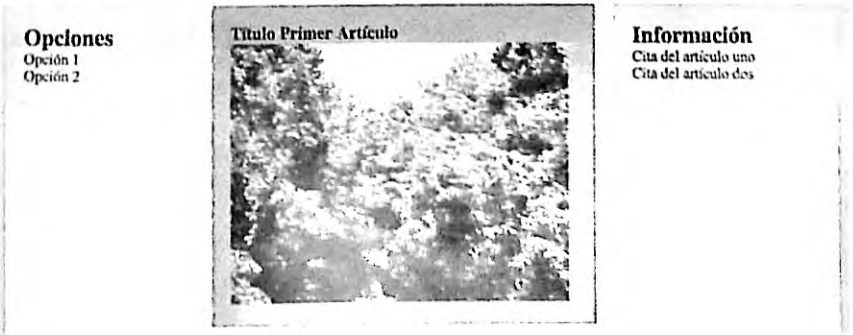

Figura 5-5: Columnas fijas a ambos lados

Hágalo usted mismo: cree un nuevo archivo HTML con el documento del Listado 5-12 y un archivo CSS llamado misestilos.css con el código del Listado 5-13. Incluya el archivo miimagen.jpg en el mismo directorio. Abra el documento en su navegador y cambie el tamaño de la ventana. Debería ver algo similar a la Figura 5-5, con las columnas a los lados siempre del mismo tamaño.

Texto

Otro aspecto del diseño que tenemos que adaptar a diferentes dispositivos es el texto. Cuando declaramos un tipo de letra con un tamaño fijo, como 14 píxeles, el texto se muestra siempre en ese tamaño, independientemente del dispositivo. En ordenadores personales puede verse bien, pero en pantallas pequeñas un texto de este tamaño será difícil de leer. Para ajustar el tamaño de la letra al dispositivo, podemos usar un tamaño de letra estándar. Este es un tamaño determinado por el navegador de acuerdo con el dispositivo en el que se ejecuta y ajustado a las preferencias del usuario. Por ejemplo, en ordenadores personales, el tamaño de la fuente por defecto es normalmente 16 píxeles. Para asegurarnos de que el texto de nuestra página es legible, podemos definir su tamaño en relación a este valor. CSS ofrece las siguientes unidades de medida con este propósito.

em—Esta unidad representa una medida relativa al tamaño de la fuente asignado al elemento. Acepta números decimales. El valor 1 es igual al tamaño actual de la fuente.

rem—Esta unidad representa una medida relativa al tamaño de la fuente asignada el elemento raíz (usualmente, el elemento **<body>**). Acepta números decimales. El valor 1 es igual al tamaño actual de la fuente.

Las fuentes se definen del mismo modo que lo hemos hecho antes, pero la unidad **px** se debe reemplazar por la unidad **em** para declarar el tamaño de la fuente relativo al tamaño de fuente actual asignado al elemento. Por ejemplo, la siguiente hoja de estilo define tamaños de letra para los elementos **<h1>** y **<p>** en nuestro documento usando unidades **em** (las reglas se han definido considerando el documento del Listado 5-6).

```
* {
  margin: 0px;
  padding: 0px;
}
section {
  float: left;
  width: 61%;
  padding: 2%;
  margin-right: 5%;
  background-color: #999999;
}
aside {
  float: left;
  width: 26%;
  padding: 2%;
  background-color: #CCCCCC;
}
.recuperar {
  clear: both;
}
```

```
article h1 {
  font: bold 2em Georgia, sans-serif;
}
aside h1 {
  font: bold 1.2em Georgia, sans-serif;
}
aside p {
  font: 1em Georgia, sans-serif;
}
```

Listado 5-14: Declarando tamaños de letra relativos

La unidad em reemplaza a la unidad px y, por lo tanto, el navegador es responsable de calcular el tamaño del texto en pixeles a partir del tamaño de la fuente actualmente asignada al elemento. Para calcular este valor, el navegador multiplica el número de em por el tamaño actual de la fuente estándar. Por ejemplo, la regla `article h1` del Listado 5-14 asigna un tamaño de 2em el elemento `<h1>` dentro de los elementos `<article>`. Esto significa que en un ordenador de escritorio el texto en estos elementos tendrá un tamaño de 32 píxeles (16 * 2).

Figura 5-6: Tamaño de letra relativo

Hágalo usted mismo: el ejemplo del Listado 5-14 asume que estamos trabajando con el documento del Listado 5-6. Reemplace las reglas en el archivo adaptabletodos.css creado para este documento con el código del Listado 5-14 y abra el documento en su navegador. Debería ver algo similar a la Figura 5-6.

La unidad em define el tamaño relativo al tamaño de fuente asignado al elemento. Esto se debe a que los elementos heredan propiedades de sus contenedores y, por lo tanto, sus tamaños de letra pueden ser diferentes del estándar establecido por el navegador. Esto significa que si modificamos el tamaño de letra del contenedor de un elemento, el tamaño de letra asignado al elemento se verá afectado. Por ejemplo, las siguiente reglas asignan un tamaño de letra al elemento `<article>` y otro al elemento `<h1>` en su interior.

```
article {
  font: bold 1.5em Georgia, sans-serif;
}
article h1 {
```

```
  font: bold 2em Georgia, sans-serif;
}
```

Listado 5-15: Declarando tamaños relativos para los elementos y sus contenedores

La primera regla asigna un tamaño de letra de `1.5em` al elemento `<article>`. Después de aplicarse esta propiedad, el contenido del elemento `<article>`, incluido el contenido del elemento `<h1>` en su interior, tendrá un tamaño de 1.5 veces el tamaño estándar (24 píxeles en un ordenador personal). El elemento `<h1>` hereda este valor y, por lo tanto, cuando se aplica la siguiente regla, el tamaño de la letra no se calcula desde el tamaño estándar sino desde el tamaño establecido por el contenedor (16 * 1.5 * 2 = 48). El resultado se muestra en la Figura 5-7.

Figura 5-7: Tamaño de letra relativo al del contenedor

Si queremos cambiar este comportamiento y forzar al navegador a calcular el tamaño de letra siempre desde el valor estándar, podemos usar las unidades `rem`. Estas unidades son relativas al elemento raíz en lugar del elemento actual y, por lo tanto, los valores siempre se multiplican por el mismo valor base.

```
article {
  font: bold 1.5rem Georgia, sans-serif;
}
article h1 {
  font: bold 2rem Georgia, sans-serif;
}
```

Listado 5-16: Declarando tamaños relativos para los elementos con unidades `rem`

Si aplicamos las reglas del Listado 5-16 al ejemplo del Listado 5-14, el navegador calcula el tamaño de la fuente a partir del valor estándar y muestra el título del artículo al doble de este tamaño, según ilustra la Figura 5-6, sin importar el tamaño declarado para el contenedor.

Lo básico: también puede usar las unidades `em` en lugar de porcentajes para definir el tamaño de los elementos. Cuando utiliza las unidades `em`, el tamaño queda determinado por el tamaño de la fuente en lugar del contenedor. Esta técnica se usa para crear diseños elásticos. Para más información, visite nuestro sitio web y siga los enlaces de este capítulo.

Imágenes

Las imágenes se muestran por defecto en su tamaño original, lo que significa que se adaptan al espacio disponible a menos que lo declaremos de forma explícita. Para convertir una imagen fija en una imagen flexible tenemos que declarar su tamaño en porcentaje. Por ejemplo, la siguiente regla configura el ancho de los elementos `<img>` dentro de los elementos `<article>` de nuestro documento con un valor de 100 % y, por lo tanto, las imágenes tendrán un ancho igual al de su contenedor.

```
article img {
  width: 100%;
}
```

Listado 5-17: Adaptando el tamaño de las imágenes

Hágalo usted mismo: agregue la regla del Listado 5-17 a la hoja de estilo creada para el ejemplo anterior y abra el documento en su navegador. Debería ver la imagen expandirse o encogerse junto con la columna a medida que cambia el tamaño de la ventana.

Al asignar un porcentaje al ancho de la imagen, se fuerza al navegador a calcular el tamaño de la imagen de acuerdo con el tamaño de su contenedor (el elemento `<article>` en nuestro documento). También podemos declarar valores menores a 100 %, pero el tamaño de la imagen siempre será proporcional al tamaño de su contenedor, lo cual significa que si el contenedor se expande, la imagen se podría presentar con un tamaño más grande que el original. Si queremos establecer los límites para expandir o encoger la imagen, tenemos que usar las propiedades `max-width` y `min-width`. Ya hemos visto estas propiedades aplicadas en el modelo de caja flexible, pero también se pueden emplear en el modelo de caja tradicional para declarar límites para elementos flexibles. Por ejemplo, si queremos que la imagen se reduzca solo cuando no hay espacio suficiente para mostrarla en su tamaño original, podemos usar la propiedad `max-width`.

```
article img {
  max-width: 100%;
}
```

Listado 5-18: Declarando un tamaño máximo para las imágenes

La regla del Listado 5-18 deja que la imagen se expanda hasta que alcanza su tamaño original. La imagen será tan ancha como su contenedor a menos que el contenedor tenga un tamaño mayor al tamaño original de la imagen.

Hágalo usted mismo: reemplace la regla del Listado 5-17 en su hoja de estilo con la regla del Listado 5-18. Abra el documento en su navegador y modifique el tamaño de la ventana. Debería ver la imagen expandirse hasta que alcance su tamaño original, tal como muestra la Figura 5-8.

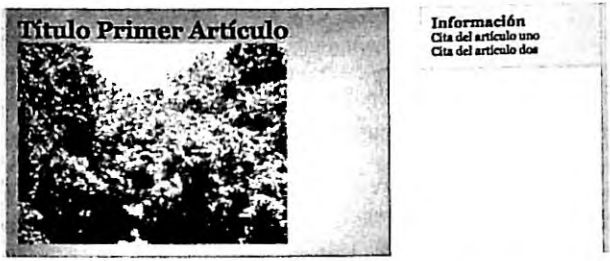

Figura 5-8: Imagen con un ancho máximo

Además de adaptar la imagen al espacio disponible, un sitio web adaptable también necesita que unas imágenes se reemplacen por otras cuando las condiciones cambian demasiado. Existen al menos dos situaciones en las que esto es necesario: cuando el espacio disponible no es suficiente para mostrar la imagen original (cuando el logo del sitio web tiene que ser reemplazado por una versión reducida, por ejemplo) o cuando la densidad de píxeles de la pantalla es diferente y necesitamos mostrar una imagen con una resolución más alta o más baja. Independientemente del motivo, HTML ofrece el siguiente elemento para seleccionar la imagen a mostrar.

<picture>—Este elemento es un contenedor que nos permite especificar múltiples fuentes para el mismo elemento `<img>`.

<source>—Este elemento define una posible fuente para el elemento `<img>`. Puede incluir el atributo `media`, para especificar la Media Query a la que la imagen estará asociada, y el atributo `srcset` para especificar la ruta de las imágenes que queremos declarar como fuentes del elemento.

El elemento `<picture>` es solo un contenedor que debe incluir uno o más elementos `<source>` para especificar las posibles fuentes de la imagen y un elemento `<img>` al final para mostrar la imagen seleccionada en pantalla. El siguiente documento ilustra cómo reemplazar el logo de un sitio web por una versión simplificada en dispositivos con pantallas pequeñas usando estos elementos.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Este texto es el título del documento</title>
  <meta charset="utf-8">
  <meta name="description" content="Este es un documento HTML5">
  <meta name="keywords" content="HTML, CSS, JavaScript">
  <meta name="viewport" content="width=device-width, initial-scale=1">
</head>
<body>
  <header>
    <picture>
      <source media="(max-width: 480px)" srcset="logoreducido.jpg">
      <img src="logo.jpg">
    </picture>
  </header>
```

```
</body>
</html>
```

Listado 5-19: Seleccionando la imagen con el elemento `<picture>`

En el documento del Listado 5-19 declaramos solo una fuente para la imagen y la asociamos a la Media Query `max-width: 480px`. Cuando el navegador lee este documento, primero procesa el elemento `<source>` y luego asigna la fuente al elemento `<img>` si el área de visualización cumple los requisitos de la Media Query. Si no se encuentran coincidencias, se muestra en su lugar la imagen especificada por el atributo `src` del elemento `<img>`. Esto significa que cada vez que el área de visualización es superior a 480 píxeles, la imagen logo.jpg se carga y se muestra en pantalla.

Figura 5-9: Logo original

Si el ancho del área de visualización es igual o menor que 480 píxeles, el navegador asigna la imagen especificada por el elemento `<source>` al elemento `<img>`, y la imagen logoreducido.jpg se muestra en pantalla.

Figura 5-10: Logo para pantallas pequeñas

Hágalo usted mismo: cree un nuevo archivo HTML con el documento del Listado 5-19. Descargue las imágenes logo.jpg y logoreducido.jpg desde nuestro sitio web y muévalas al directorio donde se encuentra el documento. Abra el documento en su navegador y cambie el tamaño de la ventana. Dependiendo del tamaño actual, debería ver algo similar a la Figura 5-9 o la Figura 5-10.

Aunque este es probablemente el escenario más común en el que una imagen debe ser reemplazada por otra de diferente tamaño o contenido, existe una condición más que debemos considerar cuando mostramos imágenes a nuestros usuarios. Desde que se introdujeron los monitores de tipo retina por parte de Apple en el año 2010, existen pantallas con una densidad más alta de píxeles. Una densidad más alta significa más píxeles por pulgada, lo que se traduce en imágenes más claras y definidas. Pero si queremos aprovechar esta característica, debemos facilitar imágenes de alta resolución en estas pantallas. Por ejemplo, una imagen de 300 píxeles de ancho por 200 píxeles de alto dentro de un área del mismo tamaño se verá bien en pantallas con una densidad normal, pero los dispositivos con alta densidad de píxeles tendrán que estirar la imagen para ocupar todos los píxeles disponibles en la misma área. Si queremos que esta imagen también se vea bien en pantallas retina con una escala de 2 (doble cantidad de píxeles por área), tenemos que reemplazarla por la misma imagen con una resolución dos veces superior (600 x 400 píxeles).

Las Media Queries contemplan esta situación con la palabra clave **resolution**. Esta palabra clave requiere que el valor se declare con un número entero y la unidad **dppx**. El valor 1 representa una resolución estándar. Valores más altos definen la escala de la pantalla. Actualmente, las pantallas retina tienen escalas de 2 y 3. El siguiente documento declara una fuente específica para pantallas con una escala de 2 (las más comunes).

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Este texto es el título del documento</title>
  <meta charset="utf-8">
  <meta name="description" content="Este es un documento HTML5">
  <meta name="keywords" content="HTML, CSS, JavaScript">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="misestilos.css">
</head>
<body>
  <section>
    <div>
      <article>
        <h1>Título Primer Artículo</h1>
        <picture>
          <source media="(resolution: 2dppx)" srcset="miimagen@2x.jpg">
          <img src="miimagen.jpg">
        </picture>
      </article>
    </div>
  </section>
  <aside>
    <div>
      <h1>Información</h1>
      <p>Cita del artículo uno</p>
      <p>Cita del artículo dos</p>
    </div>
  </aside>
  <div class="recuperar"></div>
</body>
</html>
```

***Listado 5-20:** Seleccionando imágenes para diferentes resoluciones*

Si este documento se abre en una pantalla con una densidad de píxeles estándar, el navegador muestra la imagen miimagen.jpg, pero cuando el dispositivo tiene una pantalla retina con una escala de 2, el navegador asigna la imagen miimagen@2x.jpg al elemento **<img>** y la muestra en pantalla.

Como ilustra el ejemplo del Listado 5-20, el proceso es sencillo. Tenemos que crear dos imágenes, una con una resolución normal y otra con una resolución más alta para dispositivos con pantalla retina, y luego declarar las fuentes disponibles con los elementos **<source>**. Pero esto presenta un problema. Si no especificamos el tamaño de las imágenes, estas se muestran en sus tamaños originales y, por lo tanto, el tamaño de la imagen de alta resolución será el doble del de la imagen normal. Por esta razón, cada vez que trabajamos con imágenes de baja y alta resolución, tenemos que declarar sus tamaños de forma explícita. Por ejemplo,

las siguientes reglas configuran el ancho de la imagen con un valor de 100 % del tamaño de su contenedor, lo cual hará que ambas imágenes adopten el mismo tamaño.

```
* {
  margin: 0px;
  padding: 0px;
}
section {
  float: left;
  width: 61%;
  padding: 2%;
  margin-right: 5%;
  background-color: #999999;
}
aside {
  float: left;
  width: 26%;
  padding: 2%;
  background-color: #CCCCCC;
}
.recuperar {
  clear: both;
}
article img {
  width: 100%;
}
```

***Listado 5-21:** Declarando el tamaño de una imagen de alta resolución*

Esto es lo mismo que hemos hecho con anterioridad, pero esta vez el navegador muestra imágenes de diferente resolución dependiendo de las características del dispositivo. La Figura 5-11 muestra cómo se ve el documento en una pantalla retina (hemos agregado el texto "2X" en la parte inferior del archivo miimagen@2x.jpg para diferenciar la imagen de baja resolución de la de alta resolución).

***Figura 5-11:** Imagen de alta resolución mostrada en una pantalla Retina*

Como hemos hecho en el ejemplo anterior, podemos usar la propiedad `max-width` para limitar el ancho de la imagen a su tamaño original (ver Listado 5-18), pero como esta vez

estamos usando dos imágenes de diferentes tamaños, no podemos declarar el valor de esta propiedad en porcentaje. En su lugar, tenemos que usar el ancho exacto que queremos que sea el máximo permitido. En nuestro ejemplo, la imagen en el archivo miimagen.jpg tiene un ancho original de 365 píxeles.

```
article img {
  width: 100%;
  max-width: 365px;
}
```

Listado 5-22: Declarando el máximo tamaño de una imagen de alta resolución

Ahora, sin importar la imagen que selecciona el navegador, se mostrará con un tamaño máximo de 365 píxeles.

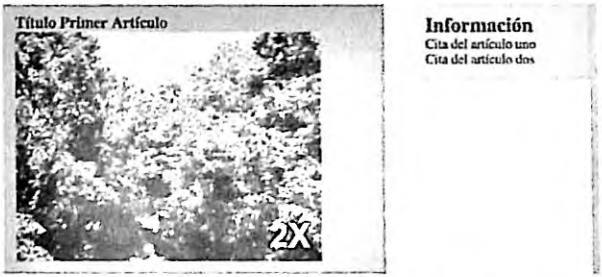

Figura 5-12: Limitaciones para imágenes de alta resolución

Hágalo usted mismo: cree un nuevo archivo HTML con el documento del Listado 5-20 y un archivo CSS llamado misestilos.css con el código del Listado 5-21. Descargue las imágenes miimagen.jpg y miimagen@2x.jpg desde nuestro sitio web y muévalas al directorio del documento. Abra el documento en su navegador. Si su ordenador incluye una pantalla Retina, debería ver la imagen con el texto "2X" en la parte inferior, tal como ilustra la Figura 5-11. Actualice la hoja de estilo con la regla del Listado 5-22 y cargue de nuevo la página. Debería ver la imagen expandirse hasta que alcanza su tamaño original (365 píxeles).

Lo básico: existe una convención que sugiere declarar los nombres de los archivos que contienen las imágenes de alta resolución con el mismo nombre que la imagen estándar seguido del carácter @, el valor de la escala, y la letra x, como en @2x o @3x. Esta es la razón por la que asignamos el nombre miimagen@2x.jpg al archivo que contiene la imagen de alta resolución.

Al igual que las imágenes introducidas con el elemento `<img>`, las imágenes de fondo también se pueden reemplazar, pero en este caso no necesitamos ningún elemento HTML porque podemos llevar a cabo todo el proceso desde CSS usando las propiedades `background` o `background-image`. Por este motivo, la imagen no se define en el

documento, sino en la hoja de estilo. El documento solo tiene que incluir el elemento al cual le asignaremos la imagen de fondo.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Este texto es el título del documento</title>
  <meta charset="utf-8">
  <meta name="description" content="Este es un documento HTML5">
  <meta name="keywords" content="HTML, CSS, JavaScript">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="misestilos.css">
</head>
<body>
  <header>
    <div id="logo"></div>
  </header>
</body>
</html>
```

Listado 5-23: Incluyendo el elemento necesario para mostrar la imagen de fondo

La cabecera de este documento contiene un elemento `<div>` identificado con el nombre `logo` que vamos a usar para mostrar el logo de nuestro sitio web. Como hemos hecho anteriormente, para cambiar el valor de una propiedad de acuerdo a diferentes condiciones, tenemos que definir el valor por defecto y luego declarar Media Queries para modificarlo cuando cambian las circunstancias.

```
* {
  margin: 0px;
  padding: 0px;
}
#logo {
  width: 275px;
  height: 60px;
  background: url("logo.jpg") no-repeat;
}
@media (max-width: 480px) {
  #logo {
    width: 60px;
    background: url("logoreducido.jpg") no-repeat;
  }
}
```

Listado 5-24: Actualizando la imagen de fondo

La hoja de estilo del Listado 5-24 define el tamaño del elemento `<div>` igual al tamaño de la imagen que queremos mostrar y luego asigna la imagen logo.jpg como su imagen de fondo (ver Figura 5-9). Si el área de visualización es de 480 píxeles o inferior, la Media Query ajusta el tamaño del elemento `<div>` para que coincida con el tamaño del logo pequeño y asigna la imagen logoreducido.jpg al fondo del elemento (ver Figura 5-10). El efecto que produce este código es exactamente el mismo que el que se logra con el elemento `<picture>` en el

documento del Listado 5-19, excepto que esta vez la imagen no la presenta el elemento `<img>`, sino que se asigna al fondo del elemento `<div>`.

Hágalo usted mismo: cree un nuevo archivo HTML con el documento del Listado 5-23 y un archivo CSS llamado misestilos.css con el código del Listado 5-24. Descargue las imágenes logo.jpg y logoreducido.jpg desde nuestro sitio web y muévalas al directorio de su documento. Abra el documento en su navegador y modifique el tamaño de la ventana. Dependiendo del tamaño actual, debería ver algo similar a la Figura 5-9 o la Figura 5-10.

Aplicación de la vida real

Crear un sitio web adaptable exige combinar todas las técnicas que hemos estudiado. Algunas se deben aplicar varias veces y, normalmente, se debe declarar más de una Media Query para adaptar el diseño a múltiples dispositivos. Pero el sitio en el que establecemos los puntos de interrupción y cómo se implementan estas técnicas depende del diseño de nuestro sitio web y del modelo de caja que implementemos. Para demostrar cómo crear un sitio web con diseño web adaptable y el modelo de caja tradicional, vamos a usar el documento del Capítulo 4, Listado 4-20. La estructura básica de este documento incluye una cabecera, una barra de navegación, dos columnas creadas con los elementos `<section>` y `<aside>`, y un pie de página. Lo primero es definir los estilos por defecto que se necesitan para transformar estos elementos en elementos flexibles.

Como hemos mencionado anteriormente, para volver a un elemento flexible en el modelo de caja tradicional, tenemos que definir sus anchos con valores en porcentaje. Los siguientes son los estilos por defecto requeridos por el documento del Capítulo 4, Listado 4-20 (estos son los estilos que se aplicarán si ninguna Media Query coincide con el tamaño del área de visualización). A diferencia de lo que hemos hecho en el Capítulo 4, esta vez los tamaños se declaran en porcentaje y usamos la propiedad `max-width` para especificar un ancho máximo para el contenido de la página.

```
* {
  margin: 0px;
  padding: 0px;
}
#cabeceralogo {
  width: 96%;
  height: 150px;
  padding: 0% 2%;
  background-color: #0F76A0;
}
#cabeceralogo > div {
  max-width: 960px;
  margin: 0px auto;
  padding-top: 45px;
}
#cabeceralogo h1 {
  font: bold 54px Arial, sans-serif;
  color: #FFFFFF;
}
#menuprincipal {
  width: 96%;
  height: 50px;
```

```
  padding: 0% 2%;
  background-color: #9FC8D9;
  border-top: 1px solid #094660;
  border-bottom: 1px solid #094660;
}
#menuprincipal > div {
  max-width: 960px;
  margin: 0px auto;
}
#menuprincipal li {
  display: inline-block;
  height: 35px;
  padding: 15px 10px 0px 10px;
  margin-right: 5px;
}
#menuprincipal li:hover {
  background-color: #6FACC6;
}
#menuprincipal a {
  font: bold 18px Arial, sans-serif;
  color: #333333;
  text-decoration: none;
}
main {
  width: 96%;
  padding: 2%;
  background-image: url("fondo.png");
}
main > div {
  max-width: 960px;
  margin: 0px auto;
}
#articulosprincipales {
  float: left;
  width: 65%;
  padding-top: 30px;
  background-color: #FFFFFF;
  border-radius: 10px;
}
#infoadicional {
  float: right;
  width: 29%;
  padding: 2%;
  background-color: #E7F1F5;
  border-radius: 10px;
}
#infoadicional h1 {
  font: bold 18px Arial, sans-serif;
  color: #333333;
  margin-bottom: 15px;
}
.recuperar {
  clear: both;
}
article {
  position: relative;
```

```
  padding: 0px 40px 20px 40px;
}
article time {
  display: block;
  position: absolute;
  top: -5px;
  left: -70px;
  width: 80px;
  padding: 15px 5px;
  background-color: #094660;
  box-shadow: 3px 3px 5px rgba(100, 100, 100, 0.7);
  border-radius: 5px;
}
.numerodia {
  font: bold 36px Verdana, sans-serif;
  color: #FFFFFF;
  text-align: center;
}
.nombredia {
  font: 12px Verdana, sans-serif;
  color: #FFFFFF;
  text-align: center;
}
article h1 {
  margin-bottom: 5px;
  font: bold 30px Georgia, sans-serif;
}
article p {
  font: 18px Georgia, sans-serif;
}
figure {
  margin: 10px 0px;
}
figure img {
  max-width: 98%;
  padding: 1%;
  border: 1px solid;
}
#pielogo {
  width: 96%;
  padding: 2%;
  background-color: #0F76A0;
}
#pielogo > div {
  max-width: 960px;
  margin: 0px auto;
  background-color: #9FC8D9;
  border-radius: 10px;
}
.seccionpie {
  float: left;
  width: 27.33%;
  padding: 3%;
}
.seccionpie h1 {
  font: bold 20px Arial, sans-serif;
}
```

```
.seccionpie p {
  margin-top: 5px;
}
.seccionpie a {
  font: bold 16px Arial, sans-serif;
  color: #666666;
  text-decoration: none;
}
```

Listado 5-25: Definiendo elementos flexibles con el modelo de caja tradicional

Las reglas del Listado 5-25 vuelven flexibles los elementos estructurales, de modo que el sitio web se adapta al espacio disponible, pero cuando la pantalla es demasiado pequeña, el diseño se rompe, algunos elementos se muestran de forma parcial y el contenido se vuelve imposible de leer, tal como muestra la Figura 5-13.

Figura 5-13: Sitio web adaptable solo con elementos flexibles

Hágalo usted mismo: cree un nuevo archivo HTML con el documento del Listado 4-20. Cree un archivo CSS llamado misestilos.css e incluya las reglas del Listado 5-25. Abra el documento en su navegador. A este momento, los elementos se expanden o encogen de acuerdo con el espacio disponible, pero la página no se ve bien cuando el área de visualización es muy pequeña (ver Figura 5-13). Agregaremos Media Queries a la hoja de estilo a continuación para corregir esta situación.

Los cambios en el diseño se tienen que introducir gradualmente. Por ejemplo, en nuestro diseño, cuando el tamaño del área de visualización es de 1120 píxeles o inferior, el elemento `<time>` que usamos para representar la fecha en la que el artículo se ha publicado se queda fuera de la ventana. Esto nos indica que nuestro diseño necesita un punto de interrupción a 1120 píxeles para mover este elemento a una posición diferente o reorganizar el contenido. En este caso, decidimos mover la fecha de vuelta dentro del área ocupada por el elemento `<article>`.

```
@media (max-width: 1120px) {
  article time {
    position: static;
    width: 100%;
```

```
    padding: 0px;
    margin-bottom: 10px;

    background-color: #FFFFFF;
    box-shadow: 0px 0px 0px;
    border-radius: 0px;
  }
  .numerodia {
    display: inline-block;
    font: bold 14px Verdana, sans-serif;
    color: #999999;
    padding-right: 5px;
  }
  .nombredia {
    display: inline-block;
    font: bold 14px Verdana, sans-serif;
    color: #999999;
  }
  article h1 {
    margin-bottom: 0px;
  }
}
```

Listado 5-26: Reposicionando el elemento `<time>`

Lo primero que tenemos que hacer para reposicionar la fecha es restaurar el modo de posicionamiento del elemento **`<time>`**. Las reglas por defecto introducidas en la hoja de estilo del Listado 5-25 le otorgan una posición absoluta al elemento para moverlo al lado derecho del área ocupada por el elemento **`<article>`**, pero cuando la pantalla no es lo suficientemente grande para mostrarlo en esa posición, tenemos que moverlo nuevamente a su ubicación natural en el documento, debajo del elemento **`<h1>`**. Esto se logra asignando el valor **`static`** a la propiedad **`position`** (**`static`** es el modo por defecto). Ahora, el elemento **`<time>`** se coloca debajo del título del artículo, pero aún tenemos que ubicar la fecha y el día en la misma línea. Para este ejemplo, decidimos convertir los elementos en elementos **`inline-block`**, por lo que se ubicarán uno al lado del otro en la misma línea (Figura 5-14).

Figura 5-14: La fecha se desplaza a la posición debajo del título con Media Queries

Hágalo usted mismo: agregue la Media Query definida en el Listado 5-26 al final de su hoja de estilo y actualice la página en su navegador. Reduzca el tamaño de la ventana para ver cómo se modifica el elemento **`<time>`**.

Otro momento en el que se debe modificar el diseño es cuando las dos columnas se vuelven demasiado pequeñas para mostrar el contenido apropiadamente. Dependiendo de las características del contenido, podemos optar por ocultarlo, reemplazarlo con un contenido diferente o reorganizar las columnas. En este caso, decidimos convertir el diseño de dos columnas en un diseño de una columna moviendo el elemento `<aside>` debajo del elemento `<section>`. Existen varias maneras de lograr este objetivo. Por ejemplo, podemos asignar el valor `none` a la propiedad `float` para prevenir que los elementos floten a los lados, o simplemente definir el ancho de los elemento con los valores 100% o `auto`, y dejar que el navegador se encargue de ubicarlos uno por línea. Para nuestro ejemplo, decidimos establecer un diseño de una sola columna cuando el área de visualización tiene un ancho de 720 píxeles o menos usando la segunda opción.

```
@media (max-width: 720px) {
  #articulosprincipales {
    width: 100%;
  }
  #infoadicional {
    width: 90%;
    padding: 5%;
    margin-top: 20px;
  }
}
```

***Listado 5-27:** Reorganizando las columnas*

Cada vez que el documento se muestre en una pantalla de un tamaño de 720 píxeles o inferior, el usuario verá el contenido organizado en una sola columna.

***Figura 5-15:** Diseño de una columna*

Hágalo usted mismo: agregue la Media Query definida en el Listado 5-27 al final de su hoja de estilo y actualice la página en su navegador. Cuando el ancho del área de visualización sea de 720 píxeles o inferior, debería ver el contenido en una sola columna, tal como ilustra la Figura 5-15.

En este momento, el documento se ve bien en ordenadores personales y tablets, pero aún debemos realizar varios cambios para adaptarlo a las pequeñas pantallas de los teléfonos móviles. Cuando esta página se muestre en un área de visualización de 480 píxeles o inferior, las opciones del menú no tendrán espacio suficiente para visualizarse en una sola línea, y el

contenido del pie de página puede que no tenga espacio suficiente para mostrarse por completo. Una manera de resolver este problema es listando los ítems uno encima del otro.

```
@media (max-width: 480px) {
  #cabeceralogo > div {
    text-align: center;
  }
  #cabeceralogo h1 {
    font: bold 46px Arial, sans-serif;
  }
  #menuprincipal {
    width: 100%;
    height: 100%;
    padding: 0%;
  }
  #menuprincipal li {
    display: block;
    margin-right: 0px;
    text-align: center;
  }
  .seccionpie {
    width: 94%;
    text-align: center;
  }
}
```

Listado 5-28: Creando un menú móvil

Con las reglas del Listado 5-28 modificamos los elementos para forzar al navegador a mostrarlos uno por línea y centrar sus contenidos. Las primeras dos reglas centran el contenido del elemento `cabeceralogo` y asignan un nuevo tamaño al título de la página para que se muestre mejor en pantallas pequeñas. A continuación, definimos el tamaño del elemento `menuprincipal` (el contenedor del menú) para que tenga el máximo ancho posible y una altura determinada por su contenido (`height: 100%`). También declaramos los elementos `<li>` dentro del elemento `menuprincipal` como elementos Block para mostrar las opciones del menú una por línea. Finalmente, los anchos de las tres secciones dentro del pie de página también se extienden para forzar al navegador a mostrarlas una por línea. La Figura 5-16 ilustra cómo afectan estos cambios a algunos de los elementos.

Figura 5-16: Menú móvil

Hágalo usted mismo: agregue la Media Query definida en el Listado 5-28 al final de su hoja de estilo y actualice la página en su navegador. Reduzca el tamaño de la ventana para ver cómo se adaptan al espacio disponible las opciones del menú y las secciones del pie de página.

Después de aplicar estas reglas, el pie de página se ve bien, pero las opciones del menú de la parte superior de la pantalla desplazan el contenido relevante hacia abajo, forzando al usuario a desplazar la página para poder verlo. Una alternativa es reemplazar el menú con un botón y mostrar las opciones solo cuando se pulsa el botón. Para este propósito, tenemos que agregar una imagen al documento que ocupará el lugar del menú cuando el ancho del área de visualización sea de 480 píxeles o inferior.

```
<nav id="menuicono">
  <img src="iconomenu.png" width="35" height="28" id="menu-img">
</nav>
<nav id="menuprincipal">
  <div>
    <ul>
      <li><a href="index.html">Principal</a></li>
      <li><a href="fotos.html">Fotos</a></li>
      <li><a href="videos.html">Videos</a></li>
      <li><a href="contacto.html">Contacto</a></li>
    </ul>
  </div>
</nav>
```

Listado 5-29: Agregando el botón del menú para pantallas pequeñas

La primera modificación que tenemos que introducir en nuestra hoja de estilo es una regla que oculta el elemento **menuicono** porque solo lo queremos mostrar en pantallas pequeñas. Existen dos formas de hacerlo: definir la propiedad **visibility** con el valor **hidden** o declarar el modo de visualización como none con la propiedad **display**. La primera opción no muestra el elemento al usuario, pero el elemento aún ocupa un espacio en la página, mientras que la segunda le dice al navegador que debe mostrar la página como si el elemento se hubiera incluido en el documento y, por lo tanto, esta última es la opción que tenemos que implementar para nuestro menú.

```
#menuicono {
  display: none;
  width: 95%;
  height: 38px;
  padding: 12px 2% 0px 3%;
  background-color: #9FC8D9;
  border-top: 1px solid #094660;
  border-bottom: 1px solid #094660;
}
```

Listado 5-30: Ocultando el botón del menú

El siguiente paso es mostrar el botón y ocultar el menú cuando el ancho del área de visualización es de 480 píxeles o inferior. Las siguientes son las modificaciones que tenemos que introducir en este punto de interrupción.

```
@media (max-width: 480px) {
  #menuprincipal {
    display: none;
    width: 100%;
    height: 100%;
    padding: 0%;
  }
  #menuprincipal li {
    display: block;
    margin-right: 0px;
    text-align: center;
  }
  #menuicono {
    display: block;
  }
  .seccionpie {
    width: 94%;
    text-align: center;
  }
  #cabeceralogo > div {
    text-align: center;
  }
}
```

Listado 5-31: Reemplazando el menú con el botón

Asignando el valor **none** a la propiedad **display** del elemento **menuprincipal** hacemos que el menú desaparezca. Si el ancho del área de visualización es de 480 píxeles o inferior, el elemento **menuicono** y su contenido se muestran en su lugar.

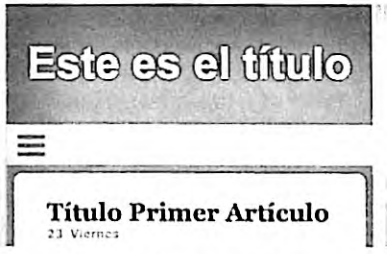

Figura 5-17: Botón del menú

Hágalo usted mismo: agregue un elemento **<nav>** identificado con el nombre **menuicono** encima del elemento **<nav>** que ya existe en su documento, como muestra el Listado 5-29. Agregue la regla del Listado 5-30 en la parte superior de su hoja de estilo. Actualice la Media Query para el punto de interrupción 480px con el código del Listado 5-31. Abra el documento en su navegador y reduzca el tamaño de la ventana. Debería ver el nuevo botón ocupando el lugar del menú cuando el ancho del área de visualización es de 480 píxeles o inferior.

En este momento, tenemos un menú que se adapta al espacio disponible, pero el botón no responde. Para mostrar el menú cuando el usuario pulsa o hace clic en el botón, tenemos que agregar a nuestro documento un programa que responda a esta acción. Estas acciones se llaman *eventos* y son controladas por código escrito en JavaScript. Estudiaremos JavaScript y eventos en el Capítulo 6, pero la tarea que debemos realizar aquí es sencilla. Tenemos que volver visible al elemento `menuprincipal` cuando el usuario pulsa el botón. La siguiente es nuestra implementación para este ejemplo.

```
<script>
  var visible = false;
  function iniciar() {
    var elemento = document.getElementById("menu-img");
    elemento.addEventListener("click", mostrarMenu);
  }
  function mostrarMenu() {
    var elemento = document.getElementById("menuprincipal");
    if (!visible) {
      elemento.style.display = "block";
      visible = true;
    } else {
      elemento.style.display = "none";
      visible = false;
    }
  }
  window.addEventListener("load", iniciar);
</script>
```

Listado 5-32: Mostrando el menú cuando se pulsa el botón

Como veremos más adelante, una forma de insertar código JavaScript dentro de un documento HTML es por medio del elemento `<script>`. Este elemento se ubica normalmente dentro de la cabecera (el elemento `<head>`), pero también se puede ubicar en cualquier otra parte del documento.

El código JavaScript, como cualquier otro código de programación, está compuesto por una serie de instrucciones que se procesan de forma secuencial. El código del Listado 5-32 primero obtiene una referencia al elemento `menu-img` y agrega una función que responderá al evento `click` de este elemento. Luego, cuando el elemento recibe el clic del usuario, el código cambia el valor de la propiedad `display` del elemento `menuprincipal` dependiendo de la condición actual. Si el menú no es visible, lo hace visible, o viceversa (explicaremos cómo funciona este código en el Capítulo 6). La Figura 5-18 muestra lo que vemos cuando se pulsa el botón.

Hágalo usted mismo: agregue el código del Listado 5-32 dentro del elemento `<head>` y debajo del elemento `<link>` en su documento. Actualice la página en su navegador. Haga clic en el botón para abrir el menú. Debería ver algo similar a la Figura 5-18.

Hasta el momento, hemos trabajado con el modelo de caja tradicional. Aunque este es el modelo preferido hoy en día debido a su compatibilidad con las versiones antiguas de los navegadores, también tenemos la opción de implementar el diseño web adaptable con el modelo de caja flexible. Para demostrar cómo desarrollar un sitio web adaptable con este modelo, vamos a usar el documento del Capítulo 4, Listado 4-52.

Figura 5-18: Menú mostrado por código JavaScript

Este ejemplo asume que hemos incluido en el documento el elemento **<nav>** agregado en el Listado 5-29 y el elemento **<script>** con el código JavaScript introducido en el Listado 5-32, pero este código se tiene que modificar para que trabaje con este modelo. El valor asignado a la propiedad **display** para hacer que el menú aparezca cuando el usuario pulsa el botón tiene que ser **flex** en lugar de **block**, porque ahora estamos trabajando con contenedores flexibles.

```
<script>
  var visible = false;
  function iniciar() {
    var elemento = document.getElementById("menu-img");
    elemento.addEventListener("click", mostrarMenu);
  }
  function mostrarMenu() {
    var elemento = document.getElementById("menuprincipal");
    if (!visible) {
      elemento.style.display = "flex";
      visible = true;
    } else {
      elemento.style.display = "none";
      visible = false;
    }
  }
  window.addEventListener("load", iniciar);
</script>
```

Listado 5-33: Mostrando el menú como un contenedor flexible

La hoja de estilo que necesitamos es muy parecida a la que usamos con el modelo de caja tradicional; la única diferencia es que tenemos que construir contenedores flexibles y declarar los elementos flexibles con la propiedad **flex**. Los siguientes son los estilos por defecto para todo el documento.

```
* {
  margin: 0px;
  padding: 0px;
}
#cabeceralogo {
  display: flex;
```

```
  justify-content: center;
  width: 96%;
  height: 150px;
  padding: 0% 2%;
  background-color: #0F76A0;
}
#cabeceralogo > div {
  flex: 1;
  max-width: 960px;
  padding-top: 45px;
}
#cabeceralogo h1 {
  font: bold 54px Arial, sans-serif;
  color: #FFFFFF;
}
#menuprincipal {
  display: flex;
  justify-content: center;
  width: 96%;
  height: 50px;
  padding: 0% 2%;
  background-color: #9FC8D9;
  border-top: 1px solid #094660;
  border-bottom: 1px solid #094660;
}
#menuprincipal > div {
  flex: 1;
  max-width: 960px;
}
#menuprincipal li {
  display: inline-block;
  height: 35px;
  padding: 15px 10px 0px 10px;
  margin-right: 5px;
}
#menuprincipal li:hover {
  background-color: #6FACC6;
}
#menuprincipal a {
  font: bold 18px Arial, sans-serif;
  color: #333333;
  text-decoration: none;
}
#menuicono {
  display: none;
  width: 95%;
  height: 38px;
  padding: 12px 2% 0px 3%;
  background-color: #9FC8D9;
  border-top: 1px solid #094660;
  border-bottom: 1px solid #094660;
}
main {
  display: flex;
  justify-content: center;
  width: 96%;
```

```
  padding: 2%;
  background-image: url("fondo.png");
}
main > div {
  display: flex;
  flex: 1;
  max-width: 960px;
}
#articulosprincipales {
  flex: 1;
  margin-right: 20px;
  padding-top: 30px;
  background-color: #FFFFFF;
  border-radius: 10px;
}
#infoadicional {
  flex: 1;
  max-width: 280px;
  padding: 2%;
  background-color: #E7F1F5;
  border-radius: 10px;
}
#infoadicional h1 {
  font: bold 18px Arial, sans-serif;
  color: #333333;
  margin-bottom: 15px;
}
article {
  position: relative;
  padding: 0px 40px 20px 40px;
}
article time {
  display: block;
  position: absolute;
  top: -5px;
  left: -70px;
  width: 80px;
  padding: 15px 5px;
  background-color: #094660;
  box-shadow: 3px 3px 5px rgba(100, 100, 100, 0.7);
  border-radius: 5px;
}
.numerodia {
  font: bold 36px Verdana, sans-serif;
  color: #FFFFFF;
  text-align: center;
}
.nombredia {
  font: 12px Verdana, sans-serif;
  color: #FFFFFF;
  text-align: center;
}
article h1 {
  margin-bottom: 5px;
  font: bold 30px Georgia, sans-serif;
}
```

```
article p {
  font: 18px Georgia, sans-serif;
}
figure {
  margin: 10px 0px;
}
figure img {
  max-width: 98%;
  padding: 1%;
  border: 1px solid;
}
#pielogo {
  display: flex;
  justify-content: center;
  width: 96%;
  padding: 2%;
  background-color: #0F76A0;
}
#pielogo > div {
  display: flex;
  flex: 1;
  max-width: 960px;
  background-color: #9FC8D9;
  border-radius: 10px;
}
.seccionpie {
  flex: 1;
  padding: 3%;
}
.seccionpie h1 {
  font: bold 20px Arial, sans-serif;
}
.seccionpie p {
  margin-top: 5px;
}
.seccionpie a {
  font: bold 16px Arial, sans-serif;
  color: #666666;
  text-decoration: none;
}
```

Listado 5-34: Diseñando un documento adaptable con el modelo de caja flexible

El diseño gráfico para este documento es el mismo que creamos anteriormente, por lo que debemos establecer los mismos puntos de interrupción. Nuevamente, cuando el ancho del área de visualización es de 1120 píxeles o inferior, tenemos que mover el elemento `<time>` debajo del título del artículo. Debido a que en ambos modelos el elemento `<time>` se posiciona con valores absolutos, esta Media Query no presenta cambio alguno.

```
@media (max-width: 1120px) {
  article time {
    position: static;
    width: 100%;
    padding: 0px;
```

```
    margin-bottom: 10px;
    background-color: #FFFFFF;
    box-shadow: 0px 0px 0px;
    border-radius: 0px;
  }
  .numerodia {
    display: inline-block;
    font: bold 14px Verdana, sans-serif;
    color: #999999;
    padding-right: 5px;
  }
  .nombredia {
    display: inline-block;
    font: bold 14px Verdana, sans-serif;
    color: #999999;
  }
  article h1 {
    margin-bottom: 0px;
  }
}
```

Listado 5-35: Moviendo el elemento `<time>`

El paso siguiente es convertir el diseño de dos columnas en un diseño de una columna cuando el ancho del área de visualización es de 720 píxeles o inferior. Debido a que ya no queremos que las columnas compartan la misma línea, sino que se muestren una encima de la otra, tenemos que declarar el contenedor como un elemento Block. Una vez que lo hemos hecho, es fácil extender los elementos hacia los lados, solo tenemos que darles un tamaño de 100 % (debido a que el elemento `<aside>` tiene por defecto un ancho máximo de 280 píxeles, también tenemos que declarar el valor de la propiedad `max-width` como 100 % para eliminar esta limitación).

```
@media (max-width: 720px) {
  main > div {
    display: block;
  }
  #articulosprincipales {
    width: 100%;
    margin-right: 0px;
  }
  #infoadicional {
    width: 90%;
    max-width: 100%;
    padding: 5%;
    margin-top: 20px;
  }
}
```

Listado 5-36: Pasando de un diseño de dos columnas a un diseño de una columna

En el último punto de interrupción tenemos que modificar la barra del menú para mostrar el botón del menú en lugar de las opciones y declarar el contenedor en el pie de página como un elemento Block para ubicar una sección sobre la otra.

```
@media (max-width: 480px) {
  #cabeceralogo > div {
    text-align: center;
  }
  #cabeceralogo h1 {
    font: bold 46px Arial, sans-serif;
  }
  #menuprincipal {
    display: none;
    width: 100%;
    height: 100%;
    padding: 0%;
  }
  #menuprincipal li {
    display: block;
    margin-right: 0px;
    text-align: center;
  }
  #menuicono {
    display: block;
  }
  #pielogo > div {
    display: block;
  }
  .seccionpie {
    width: 94%;
    text-align: center;
  }
}
```

***Listado 5-37:** Adaptando el menú y el pie de página*

Con el código del Listado 5-37, la hoja de estilo está lista. El diseño final es exactamente igual que el que logramos con el modelo de caja tradicional, pero esta vez usando el modelo de caja flexible. El modelo de caja flexible es una gran mejora con respecto al modelo de caja tradicional y puede simplificar la creación de sitios web adaptables, permitiéndonos modificar el orden en el que se presentan los elementos y facilitando la combinación de elementos de tamaño flexible y fijos, aunque no es compatible con todos los navegadores del mercado. Algunos desarrolladores ya utilizan este modelo o implementan algunas de sus propiedades, pero la mayoría de los sitios web aún se desarrollan con el modelo de caja tradicional.

Hágalo usted mismo: cree un nuevo archivo HTML con el documento del Listado 4-52 (vea el modelo de caja flexible en el Capítulo 4). Agregue el elemento `<nav>` introducido en el Listado 5-29 y el elemento `<script>` del Listado 5-33 al documento tal como hemos explicado en el ejemplo anterior. Cree un nuevo archivo CSS llamado misestilos.css y copie los códigos de los Listados 5-34, 5-35, 5-36 y 5-37 en su interior. Descargue los archivos miimagen.jpg e iconomenu.png desde nuestro sitio web y muévalos al directorio de su documento. Abra el documento en su navegador y cambie el tamaño de la ventana para ver cómo se adaptan los elementos al espacio disponible.

Capítulo 6
JavaScript

6.1 Introducción a JavaScript

HTML y CSS incluyen instrucciones para indicar al navegador cómo debe organizar y visualizar un documento y su contenido, pero la interacción de estos lenguajes con el usuario y el sistema se limita solo a un grupo pequeño de respuestas predefinidas. Podemos crear un formulario con campos de entrada, controles y botones, pero HTML solo provee la funcionalidad necesaria para enviar la información introducida por el usuario al servidor o para limpiar el formulario. Algo similar pasa con CSS; podemos construir instrucciones (reglas) con seudoclases como `:hover` para aplicar un grupo diferente de propiedades cuando el usuario mueve el ratón sobre un elemento, pero si queremos realizar tareas personalizadas, como modificar los estilos de varios elementos al mismo tiempo, debemos cargar una nueva hoja de estilo que ya presente estos cambios. Con el propósito de alterar elementos de forma dinámica, realizar operaciones personalizadas, o responder al usuario y a cambios que ocurren en el documento, los navegadores incluyen un tercer lenguaje llamado *JavaScript*.

JavaScript es un lenguaje de programación que se usa para procesar información y manipular documentos. Al igual que cualquier otro lenguaje de programación, JavaScript provee instrucciones que se ejecutan de forma secuencial para indicarle al sistema lo que queremos que haga (realizar una operación aritmética, asignar un nuevo valor a un elemento, etc.). Cuando el navegador encuentra este tipo de código en nuestro documento, ejecuta las instrucciones al momento y cualquier cambio realizado en el documento se muestra en pantalla.

Implementando JavaScript

Siguiendo el mismo enfoque que CSS, el código JavaScript se puede incorporar al documento mediante tres técnicas diferentes: el código se puede insertar en un elemento por medio de atributos (En línea), incorporar al documento como contenido del elemento `<script>` o cargar desde un archivo externo. La técnica En línea aprovecha atributos especiales que describen un evento, como un clic del ratón. Para lograr que un elemento responda a un evento usando esta técnica, todo lo que tenemos que hacer es agregar el atributo correspondiente con el código que queremos que se ejecute.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>JavaScript</title>
</head>
<body>
  <section>
    <p onclick="alert('Hizo clic!')">Clic aquí</p>
    <p>No puede hacer clic aquí</p>
  </section>
```

```
</body>
</html>
```

Listado 6-1: Definiendo JavaScript en línea

El atributo `onclick` agregado al elemento `<p>` del Listado 6-1 dice algo similar a «cuando alguien hace clic en este elemento, ejecutar este código», y el código es (en este caso) la instrucción `alert()`. Esta es una instrucción predefinida en JavaScript llamada *función*. Lo que esta función hace es mostrar una ventana emergente con el valor provisto entre paréntesis. Cuando el usuario hace clic en el área ocupada por el elemento `<p>`, el navegador ejecuta la función `alert()` y muestra una ventana emergente en la pantalla con el mensaje «Hizo clic!».

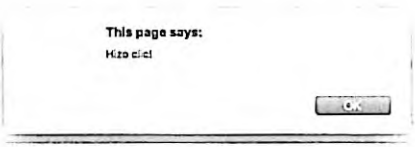

Figura 6-1: Ventana emergente generada por la función `alert()`

Hágalo usted mismo: cree un nuevo archivo HTML con el código del Listado 6-1. Abra el documento en su navegador y haga clic en el texto «Clic Aquí» (el atributo `onclick` afecta a todo el elemento, no solo al texto, por lo que también puede hacer clic en el resto del área ocupada por el elemento para ejecutar el código). Debería ver una ventana emergente con el mensaje «Hizo clic!», tal como muestra la Figura 6-1.

Lo básico: JavaScript incluye múltiples funciones predefinidas y también permite crear funciones personalizadas. Estudiaremos cómo trabajar con funciones y funciones predefinidas más adelante en este capítulo.

El atributo `onclick` es parte de una serie de atributos provistos por HTML para responder a eventos. La lista de atributos disponibles es extensa, pero se pueden organizar en grupos dependiendo de sus propósitos. Por ejemplo, los siguientes son los atributos más usados asociados con el ratón.

onclick—Este atributo responde al evento `click`. El evento se ejecuta cuando el usuario hace clic con el botón izquierdo del ratón. HTML ofrece otros dos atributos similares llamados `ondblclick` (el usuario hace doble clic con el botón izquierdo del ratón) y `oncontextmenu` (el usuario hace clic con el botón derecho del ratón).

onmousedown—Este atributo responde al evento `mousedown`. Este evento se desencadena cuando el usuario pulsa el botón izquierdo o el botón derecho del ratón.

onmouseup—Este atributo responde al evento `mouseup`. El evento se desencadena cuando el usuario libera el botón izquierdo del ratón.

onmouseenter—Este atributo responde al evento `mouseenter`. Este evento se desencadena cuando el ratón se introduce en el área ocupada por el elemento.

onmouseleave—Este atributo responde al evento `mouseleave`. Este evento se desencadena cuando el ratón abandona el área ocupada por el elemento.

onmouseover—Este atributo responde al evento `mouseover`. Este evento se desencadena cuando el ratón se mueve sobre el elemento o cualquiera de sus elementos hijos.

onmouseout—Este atributo responde al evento `mouseout`. El evento se desencadena cuando el ratón abandona el área ocupada por el elemento o cualquiera de sus elementos hijos.

onmousemove—Este atributo responde al evento `mousemove`. Este evento se desencadena cada vez que el ratón se encuentra sobre el elemento y se mueve.

onwheel—Este atributo responde al evento `wheel`. Este evento se desencadena cada vez que se hace girar la rueda del ratón.

Los siguientes son los atributos disponibles para responder a eventos generados por el teclado. Estos tipos de atributos se aplican a elementos que aceptan una entrada del usuario, como los elementos `<input>` y `<textarea>`.

onkeypress—Este atributo responde al evento `keypress`. Este evento se desencadena cuando se activa el elemento y se pulsa una tecla.

onkeydown—Este atributo responde al evento `keydown`. Este evento se desencadena cuando se activa el elemento y se pulsa una tecla.

onkeyup—Este atributo responde al evento `keyup`. Este evento se desencadena cuando se activa el elemento y se libera una tecla.

También contamos con otros dos atributos importantes asociados al documento:

onload—Este atributo responde al evento `load`. El evento se desencadena cuando un recurso termina de cargarse.

onunload—Este atributo responde al evento `unload`. Este evento se desencadena cuando un recurso termina de cargarse.

Los atributos de evento se incluyen en un elemento dependiendo de cuándo queremos que se ejecute el código. Si queremos responder al clic del ratón, tenemos que incluir el atributo `onclick`, como hemos hecho en el Listado 6-1, pero si queremos iniciar un proceso cuando el puntero del ratón pasa sobre un elemento, tenemos que incluir los atributos `onmouseover` u `onmousemove`. Debido a que en un elemento pueden ocurrir varios eventos en algunos casos al mismo tiempo, podemos declarar más de un atributo por cada elemento. Por ejemplo, el siguiente documento incluye un elemento `<p>` con dos atributos, `onclick` y `onmouseout`, que incluyen sus propios códigos JavaScript. Si el usuario hace clic en el elemento, se muestra una ventana emergente con el mensaje «Hizo clic!», pero si el usuario mueve el ratón fuera del área ocupada por el elemento, se muestra una ventana emergente diferente con el mensaje «No me abandone!».

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>JavaScript</title>
</head>
```

```
<body>
  <section>
    <p onclick="alert('Hizo clic!')" onmouseout="alert('No me
abandone!')">Clic aquí</p>
  </section>
</body>
</html>
```

Listado 6-2: Implementando múltiples atributos de evento

Hágalo usted mismo: actualice su archivo HTML con el código del Listado 6-2. Abra el documento en su navegador y mueva el ratón sobre el área ocupada por el elemento **<p>**. Si mueve el ratón fuera del área, debería ver una ventana emergente con el mensaje «No me abandone! ».

Los eventos no solo los produce el usuario, sino también el navegador. Un evento útil desencadenado por el navegador es **load**. Este evento se desencadena cuando se ha terminado de cargar un recurso y, por lo tanto, se utiliza frecuentemente para ejecutar código JavaScript después de que el navegador ha cargado el documento y su contenido.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>JavaScript</title>
</head>
<body onload="alert('Bienvenido!')">
  <section>
    <h1>Mi Sitio Web</h1>
    <p>Bienvenido a mi sitio web</p>
  </section>
</body>
</html>
```

Listado 6-3: Respondiendo al evento `load`

El documento del Listado 6-3 muestra una ventana emergente para dar la bienvenida al usuario después de que se ha cargado completamente. El navegador primero carga el contenido del documento y cuando termina, llama a la función **alert()** y muestra el mensaje en la pantalla.

IMPORTANTE: los eventos son críticos en el desarrollo web. Además de los estudiados en este capítulo, hay docenas de eventos disponibles para controlar una variedad de procesos, desde reproducir un vídeo hasta controlar el progreso de una tarea. Estudiaremos eventos más adelante en este capítulo e introduciremos el resto de los eventos disponibles en situaciones más prácticas en capítulos posteriores.

Lo básico: cuando pruebe el código del Listado 6-3 en su navegador, verá que la ventana emergente se muestra antes de que el contenido del documento aparezca en la pantalla. Esto se debe a que el documento se carga en una estructura interna de objetos llamada DOM y luego se reconstruye en la pantalla desde estos objetos. Estudiaremos la estructura DOM y cómo acceder a los elementos HTML desde JavaScript más adelante en este capítulo.

Los atributos de evento son útiles cuando queremos probar código o implementar una función de inmediato, pero no son apropiados para aplicaciones importantes. Para trabajar con códigos extensos y personalizar las funciones, tenemos que agrupar el código con el elemento `<script>`. El elemento `<script>` actúa igual que el elemento `<style>` para CSS, organizando el código en un solo lugar y afectando al resto de los elementos en el documento usando referencias.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>JavaScript</title>
  <script>
    alert('Bienvenido!');
  </script>
</head>
<body>
  <section>
    <p>Hola</p>
  </section>
</body>
</html>
```

Listado 6-4: Código JavaScript introducido en el documento

El elemento `<script>` y su contenido se pueden ubicar en cualquier parte del documento, pero normalmente se introducen dentro de la cabecera, como hemos hecho en este ejemplo. De esta manera, cuando el navegador carga el archivo, lee el contenido del elemento `<script>`, ejecuta el código al instante, y luego continúa procesando el resto del documento.

Hágalo usted mismo: actualice su archivo HTML con el código del Listado 6-4 y abra el documento en su navegador. Debería ver una ventana emergente con el mensaje «Bienvenido!» tan pronto como se carga el documento. Debido a que la función `alert()` detiene la ejecución del código, el contenido del documento no se muestra en la pantalla hasta que pulsamos el botón OK.

Introducir JavaScript en el documento con el elemento `<script>` puede resultar práctico cuando tenemos un grupo pequeño de instrucciones, pero el código JavaScript crece con rapidez en aplicaciones profesionales. Si usamos el mismo código en más de un documento, tendremos que mantener diferentes versiones del mismo programa y los navegadores tendrán que descargar el mismo código una y otra vez con cada documento solicitado por el usuario. Una alternativa es introducir el código JavaScript en un archivo externo y luego cargarlo desde los documentos que lo requieren. De esta manera, solo los documentos que necesitan ese grupo de instrucciones deberán incluir el archivo, y el navegador tendrá que descargar el archivo una sola vez (los navegadores mantienen los archivos en un caché en el ordenador del usuario en caso de que sean requeridos más adelante por otros documentos del mismo sitio web). Para este propósito, el elemento `<script>` incluye el atributo `src`. Con este atributo, podemos declarar la ruta al archivo JavaScript y escribir todo nuestro código dentro de este archivo.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>JavaScript</title>
  <script src="micodigo.js"></script>
</head>
<body>
  <section>
    <p>Hola</p>
  </section>
</body>
</html>
```

Listado 6-5: Introduciendo código JavaScript desde un archivo externo

El elemento **<script>** del Listado 6-5 carga el código JavaScript desde una archivo llamado micodigo.js. A partir de ahora, podemos insertar este archivo en cada documento de nuestro sitio web y reusar el código cada vez que lo necesitemos.

Al igual que los archivos HTML y CSS, los archivos JavaScript son simplemente archivos de texto que podemos crear con cualquier editor de texto o los editores profesionales que recomendamos en el Capítulo 1, como Atom (www.atom.io). A estos tipos de archivos se les puede asignar cualquier nombre, pero por convención tienen que tener la extensión .js. El archivo debe contener el código JavaScript exactamente según se declara entre las etiquetas **<script>**. Por ejemplo, el siguiente es el código que tenemos que introducir en el archivo micodigo.js para reproducir el ejemplo anterior.

```
alert("Bienvenido!");
```

Listado 6-6: Creando un archivo JavaScript (micodigo.js)

Hágalo usted mismo: actualice su archivo HTML con el código del Listado 6-5. Cree un nuevo archivo con el nombre micodigo.js y el código JavaScript del Listado 6-6. Abra el documento en su navegador. Debería ver una ventana emergente con el mensaje «Bienvenido!» tan pronto como se carga el documento.

Lo básico: en JavaScript se recomienda finalizar cada instrucción con un punto y coma para asegurarnos de que el navegador no tenga ninguna dificultad al identificar el final de cada instrucción. El punto y coma se puede ignorar, pero nos puede ayudar a evitar errores cuando el código está compuesto por múltiples instrucciones, como ocurre frecuentemente.

IMPORTANTE: además del atributo **src**, el elemento **<script>** también puede incluir los atributos **async** y **defer**. Estos son atributos booleanos que indican cómo y cuándo se debe ejecutar el código. Si el atributo **async** se declara, el código se ejecuta de forma asíncrona (mientras se procesa el resto del documento). Si el atributo **defer** se declara en su lugar, el código se ejecuta después de que el documento completo se haya procesado.

Variables

Por supuesto, JavaScript es algo más que ventanas emergentes mostrando mensajes para alertar al usuario. El lenguaje puede realizar numerosas tareas, desde calcular algoritmos complejos hasta procesar el contenido de un documento. Cada una de estas tareas involucra la manipulación de valores, y esta es la razón por la que la característica más importante de JavaScript, al igual que cualquier otro lenguaje de programación, es la capacidad de almacenar datos en memoria.

La memoria de un ordenador o dispositivo móvil es como un panal de abejas con millones y millones de celdas consecutivas en las que se almacena información. Estas celdas tienen un espacio limitado y, por lo tanto, es necesaria la combinación de múltiples celdas para almacenar grandes cantidades de datos. Debido a la complejidad de esta estructura, los lenguajes de programación incorporan el concepto de *variables* para facilitar la identificación de cada valor almacenado en memoria. Las variables son simplemente nombres asignados a una celda o un grupo de celdas donde se van a almacenar los datos. Por ejemplo, si almacenamos el valor 5, tenemos que saber en qué parte de la memoria se encuentra para poder leerlo más adelante. La creación de una variable nos permite identificar ese espacio de memoria con un nombre y usar ese nombre más adelante para leer el valor o reemplazarlo por otro.

Las variables en JavaScript se declaran con la palabra clave **var** seguida del nombre que queremos asignarle. Si queremos almacenar un valor en el espacio de memoria asignado por el sistema a la variable, tenemos que incluir el carácter = (igual) seguido del valor, como en el siguiente ejemplo.

```
var minumero = 2;
```

Listado 6-7: Declarando una variable en JavaScript

La instrucción del Listado 6-7 crea la variable **minumero** y almacena el valor 2 en el espacio de memoria reservado por el sistema para la misma. Cuando se ejectua este código, el navegador reserva un espacio en memoria, almacena el número 2 en su interior, crea una referencia a ese espacio, y finalmente asigna esta referencia al nombre **minumero**.

Después de asignar el valor a la variable, cada vez que se referencia esta variable (se usa el nombre **minumero**), el sistema lee la memoria y devuelve el número 2, tal como ilustra el siguiente ejemplo.

```
var minumero = 2;
alert(minumero);
```

Listado 6-8: Usando el contenido de una variable

Como ya hemos mencionado, las instrucciones en un programa JavaScript las ejecuta el navegador una por una en secuencia. Por lo tanto, cuando el navegador lee el código del Listado 6-8, ejecuta las instrucciones de arriba abajo. La primera instrucción le pide al navegador que cree una variable llamada **minumero** y le asigne el valor 2. Después de completar esta tarea, el navegador ejecuta la siguiente instrucción de la lista. Esta instrucción le pide al navegador que muestre una ventana emergente con el valor actual almacenado en la variable **minumero**.

Figura 6-2: Ventana emergente mostrando el valor de una variable

Hágalo usted mismo: actualice su archivo micodigo.js con el código del Listado 6-8. Abra el documento del Listado 6-5 en su navegador. Debería ver una ventana emergente con el valor 2.

Las variables se denominan así porque sus valores no son constantes. Podemos cambiar sus valores cada vez que lo necesitemos, y esa es, de hecho, su característica más importante.

```
var minumero = 2;
minumero = 3;
alert(minumero);
```

Listado 6-9: Asignando un nuevo valor a la variable

En el Listado 6-9, después de que se declara la variable, le es asignado un nuevo valor. Ahora, la función `alert()` muestra el número 3 (la segunda instrucción reemplaza el valor 2 por el valor 3). Cuando asignamos un nuevo valor a una variable, no tenemos la necesidad de declarar la palabra clave `var`, solo se requieren el nombre de la variable y el carácter =.

El valor almacenado en una variable se puede asignar a otra. Por ejemplo, el siguiente código crea dos variables llamadas `minumero` y `tunumero`, y asigna el valor almacenado en la variable `minumero` a la variable `tunumero`. El valor mostrado en la pantalla es el número 2.

```
var minumero = 2;
var tunumero = minumero;
alert(tunumero);
```

Listado 6-10: Asignando el valor de una variable a otra variable

En una situación más práctica, probablemente usaríamos el valor de la variable para ejecutar una operación y asignar el resultado de vuelta a la misma variable.

```
var minumero = 2;
minumero = minumero + 1;   // 3
alert(minumero);
```

Listado 6-11: Realizando una operación con el valor almacenado en una variable

En este ejemplo, el valor 1 se agrega al valor actual de `minumero` y el resultado se asigna a la misma variable. Esto es lo mismo que sumar 2 + 1, con la diferencia de que cuando usamos una variable en lugar de un número, su valor puede cambiar en cualquier momento.

Lo básico: los caracteres al final de la segunda instrucción se consideran comentarios y, por lo tanto, no se procesan como parte de la instrucción. Los comentarios se pueden agregar al código como referencias o recordatorios para el desarrollador. Se pueden declarar usando dos barras oblicuas para comentarios de una línea (`// comentario`) o combinando una barra oblicua con un asterisco para crear comentarios de varias líneas (`/* comentario */`). Todo lo que se encuentra a continuación de las dos barras o entre los caracteres `/*` y `*/` el navegador lo ignora.

Además del operador +, JavaScript también incluye los operadores - (resta), * (multiplicación), / (división), y % (módulo). Estos operadores se pueden usar en una operación sencilla entre dos valores o combinados con múltiples valores para realizar operaciones aritméticas más complejas.

```
var minumero = 2;
minumero = minumero * 25 + 3;  // 53
alert(minumero);
```

Listado 6-12: Realizando operaciones complejas

Las operaciones aritméticas se ejecutan siguiendo un orden de prioridad determinado por los operadores. La multiplicación y la división tienen prioridad sobre la adición y la substracción. Esto significa que las multiplicaciones y divisiones se calcularán antes que las sumas y restas. En el ejemplo del Listado 6-12, el valor actual de la variable `minumero` se multiplica por 25, y luego el valor 3 se suma al resultado. Si queremos controlar la precedencia, podemos aislar las operaciones con paréntesis. Por ejemplo, el siguiente código realiza la adición primero y luego la multiplicación, lo que genera un resultado diferente.

```
var minumero = 2;
minumero = minumero * (25 + 3);  // 56
alert(minumero);
```

Listado 6-13: Controlando la precedencia en la operación

Realizar una operación en el valor actual de una variable y asignar el resultado de vuelta a la misma variable es muy común en programación. JavaScript ofrece los siguientes operadores para simplificar esta tarea.

- `++` es una abreviatura de la operación `variable = variable + 1`.
- `--` es una abreviatura de la operación `variable = variable - 1`.
- `+=` es una abreviatura de la operación `variable = variable + number`.
- `-=` es una abreviatura de la operación `variable = variable - number`.
- `*=` es una abreviatura de la operación `variable = variable * number`.
- `/=` es una abreviatura de la operación `variable = variable / number`.

Con estos operadores, podemos realizar operaciones en los valores de una variable y asignar el resultado de vuelta a la misma variable. Un uso común de estos operadores es el de

crear contadores que incrementan o disminuyen el valor de una variable en una o más unidades. Por ejemplo, el operador ++ suma el valor 1 al valor actual de la variable cada vez que se ejecuta la instrucción.

```
var minumero = 0;
minumero++;
alert(minumero);  // 1
```

Listado 6-14: Incrementando el valor de una variable

Si el valor de la variable se debe incrementar más de una unidad, podemos usar el operador +=. Este operador suma el valor especificado en la instrucción al valor actual de la variable y almacena el resultado de vuelta en la misma variable.

```
var minumero = 0;
minumero += 5;
alert(minumero);  // 5
```

Listado 6-15: Incrementando el valor de una variable en un valor específico

El proceso generado por el código del Listado 6-15 es sencillo. Después de asignar el valor 0 a la variable **minumero**, el sistema lee la segunda instrucción, obtiene el valor actual de la variable, le suma el valor 5 y almacena el resultado de vuelta en **minumero**.

Una operación interesante que aún no hemos implementado es el operador módulo. Este operador devuelve el resto de una división entre dos números.

```
var minumero = 11 % 3;  // 2
alert(minumero);
```

Listado 6-16: Calculando el resto de una división

La operación asignada a la variable **minumero** del Listado 6-16 produce el resultado 2. El sistema divide 11 por 3 y encuentra el cociente 3. Luego, para obtener el resto, calcula 11 menos la multiplicación de 3 por el cociente (11 - (3 * 3) = 2).

El operador módulo se usa frecuentemente para determinar si un valor es par o impar. Si calculamos el resto de un número entero dividido por 2, obtenemos un resultado que indica si el número es par o impar. Si el número es par, el resto es 0, pero si el número es impar, el resto es 1 (o -1 para valores negativos).

```
var minumero = 11;
alert(minumero % 2);  // 1
```

Listado 6-17: Determinando la paridad de un número

El código del Listado 6-17 calcula el resto del valor actual de la variable **minumero** dividido por 2. El valor que devuelve es 1, lo que significa que el valor de la variable es impar.

En este ejemplo ejecutamos la operación entre los paréntesis de la función `alert()`. Cada vez que una operación se incluye dentro de una instrucción, el navegador primero calcula la operación y luego ejecuta la instrucción con el resultado, por lo que una operación puede ser provista cada vez que se requiere un valor.

Hágalo usted mismo: actualice su archivo micodigo.js con el ejemplo que quiere probar y abra el documento en su navegador. Reemplace los valores y realice operaciones más complejas para ver los diferentes resultados producidos por JavaScript y así familiarizarse con los operadores del lenguaje.

Cadenas de texto

En los anteriores ejemplos hemos almacenado números, pero las variables también se pueden usar para almacenar otros tipos de valores, incluido texto. Para asignar texto a una variable, tenemos que declararlo entre comillas simples o dobles.

```
var mitexto = "Hola Mundo!";
alert(mitexto);
```

Listado 6-18: Asignando una cadena de caracteres a una variable

El código del Listado 6-18 crea una variable llamada `mitexto` y le asigna una cadena de caracteres. Cuando se ejecuta la primera instrucción, el sistema reserva un espacio de memoria lo suficientemente grande como para almacenar la cadena de caracteres, crea la variable, y almacena el texto. Cada vez que leemos la variable `mitexto`, recibimos en respuesta el texto «Hola Mundo!» (sin las comillas).

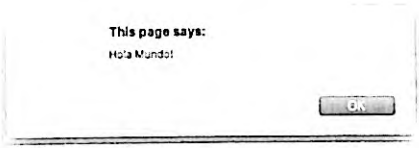

Figura 6-3: Ventana emergente mostrando una cadena de caracteres

El espacio reservado en memoria por el sistema para almacenar la cadena de caracteres depende del tamaño del texto (cuántos caracteres contiene), pero el sistema está preparado para ampliar este espacio si luego se asignan valores más extensos a la variable. Una situación común en la cual se asignan textos más extensos a la misma variable es cuando agregamos más caracteres al comienzo o al final del valor actual de la variable. El texto se puede concatenar con el operador +.

```
var mitexto = "Mi nombre es ";
mitexto = mitexto + "Juan";
alert(mitexto);
```

Listado 6-19: Concatenando texto

El código del Listado 6-19 agrega el texto "Juan" al final del texto "Mi nombre es ". El valor final de la variable **mitexto** es "Mi nombre es Juan". Si queremos agregar el texto al comienzo, solo tenemos que invertir la operación.

```
var mitexto = "Juan";
mitexto = "Mi nombre es " + mitexto;
alert(mitexto);
```

Listado 6-20: Agregando texto al comienzo del valor

Si en lugar de texto intentamos concatenar una cadena de caracteres con un número, el número se convierte en una cadena de caracteres y se agrega al valor actual. El siguiente código produce la cadena de caracteres "El número es 3".

```
var mitexto = "El número es " + 3;
alert(mitexto);
```

Listado 6-21: Concatenando texto con números

Este procedimiento es importante cuando tenemos una cadena de caracteres que contiene un número y queremos agregarle otro número. Debido a que JavaScript considera el valor actual como una cadena de caracteres, el número también se convierte en una cadena de caracteres y los valores no se suman.

```
var mitexto = "20" + 3;
alert(mitexto);  // "203"
```

Listado 6-22: Concatenando números

Lo básico: el resultado del código del Listado 6-22 no es 23, sino la cadena de caracteres "203". El sistema convierte el número 3 en una cadena de caracteres y concatena las cadenas en lugar de sumar los números. Más adelante aprenderemos cómo extraer números de una cadena de caracteres para poder realizar operaciones aritméticas con estos valores.

Las cadenas de caracteres pueden contener cualquier carácter que queramos, y esto incluye comillas simples o dobles. Si las comillas en el texto son diferentes a las comillas usadas para definir la cadena de caracteres, estas se tratan como cualquier otro carácter, pero si las comillas son las mismas, el sistema no sabe dónde termina el texto. Para resolver este problema, JavaScript ofrece el carácter de escape \. Por ejemplo, si la cadena de caracteres se ha declarado con comillas simples, tenemos que escapar las comillas simples dentro del texto.

```
var mitexto = 'El \'libro\' es interesante';
alert(mitexto);  // "El 'libro' es interesante"
```

Listado 6-23: Escapando caracteres

JavaScript ofrece varios caracteres de escape con diferentes propósitos. Los que se utilizan con más frecuencia son `\n` para generar una nueva línea y `\r` para devolver el cursor al comienzo de la línea. Generalmente, estos dos caracteres se implementan en conjunto para dividir el texto en múltiples líneas, tal como muestra el siguiente ejemplo.

```
var mitexto = "Felicidad no es hacer lo que uno quiere\r\n";
mitexto = mitexto + "sino querer lo que uno hace."
alert(mitexto);
```

Listado 6-24: Generando nuevas líneas de texto

El código del Listado 6-24 comienza asignando una cadena de caracteres a la variable `mitexto` que incluye los caracteres de escape `\r\n`. En la segunda instrucción, agregamos otro texto al final del valor actual de la variable, pero debido a los caracteres de escape, estos dos textos se muestran dentro de la ventana emergente en diferentes líneas.

Lo básico: en JavaScript las cadenas de caracteres se declaran como objetos y, por lo tanto, incluyen métodos para realizar operaciones en sus caracteres. Estudiaremos objetos, los objetos `String`, y cómo implementar sus métodos más adelante en este capítulo.

Booleanos

Otro tipo de valores que podemos almacenar en variables son los booleanos. Las variables booleanas pueden contener solo dos valores: `true` (verdadero) o `false` (falso). Estas variables son particularmente útiles cuando solo necesitamos determinar el estado actual de una condición. Por ejemplo, si nuestra aplicación necesita saber si un valor insertado en el formulario es válido o no, podemos informar de esta condición al resto del código con una variable booleana.

```
var valido = true;
alert(valido);
```

Listado 6-25: Declarando una variable booleana

El propósito de estas variables es el de simplificar el proceso de identificación del estado de una condición. Si usamos un número entero para indicar un estado, deberemos recordar qué números decidimos usar para representar los estados válido y no válido. Usando valores booleanos en su lugar, solo tenemos que comprobar si el valor es igual a `true` o `false`.

IMPORTANTE: los valores booleanos son útiles cuando los usamos junto con instrucciones que nos permiten realizar una tarea o tareas repetitivas de acuerdo a una condición. Estudiaremos las condicionales y los bucles más adelante en este capítulo.

Arrays

Las variables también pueden almacenar varios valores al mismo tiempo en una estructura llamada *array*. Los arrays se pueden crear usando una sintaxis simple que incluye los valores

separados por comas dentro de corchetes. Los valores se identifican luego mediante un índice, comenzando desde 0 (cero).

```
var miarray = ["rojo", "verde", "azul"];
alert(miarray[0]);  // "rojo"
```

Listado 6-26: Creando arrays

En el Listado 6-26, creamos un array llamado **miarray** con tres valores, las cadenas de caracteres "rojo", "verde" y "azul". JavaScript asigna automáticamente el índice **0** al primer valor, **1** al segundo, y **2** al tercero. Para leer estos datos, tenemos que mencionar el índice del valor entre corchetes después del nombre de la variable. Por ejemplo, para obtener el primer valor de **miarray**, tenemos que escribir la instrucción **miarray[0]**, como hemos hecho en nuestro ejemplo.

La función **alert()** puede mostrar no solo valores independientes, sino arrays completos. Si queremos ver todos los valores incluidos en el array, solo tenemos que especificar el nombre del array.

```
var miarray = ["rojo", "verde", "azul"];
alert(miarray);  // "rojo,verde,azul"
```

Listado 6-27: Mostrando los valores del array

Los arrays, al igual que cualquier otra variable, pueden contener cualquier tipo de valor que deseemos. Por ejemplo, podemos crear un array como el del Listado 6-27 combinando números y cadenas de caracteres.

```
var miarray = ["rojo", 32, "HTML5 es genial!"];
alert(miarray[1]);
```

Listado 6-28: Almacenando diferentes tipos de valores

Hágalo usted mismo: reemplace el código en su archivo micodigo.js por el código del Listado 6-28 y abra el documento del Listado 6-5 en su navegador. Cambie el índice provisto en la función **alert()** para mostrar cada valor en el array (recuerde que los índices comienzan desde 0).

Si intentamos leer un valor en un índice que aún no se ha definido, JavaScript devuelve el valor **undefined** (indefinido) Este valor lo usa el sistema para informar de que el valor que estamos intentando acceder no existe, pero también podemos asignarlo a un array cuando aún no contamos con el valor para esa posición.

```
var miarray = ["rojo", undefined, 32];
alert(miarray[1]);
```

Listado 6-29: Declarando valores indefinidos

Otra manera mejor de indicarle al sistema que no existe un valor disponible en un momento para un índice del array es usando otro valor especial llamado **null** (nulo). La diferencia entre los valores **undefined** y **null** es que **undefined** indica que la variable fue declarada pero ningún valor le fue asignado, mientras que **null** indica que existe un valor, pero es nulo.

```
var miarray = ["rojo", 32, null];
alert(miarray[2]);
```

Listado 6-30: Declarando el valor **null**

Por supuesto, también podemos realizar operaciones en los valores de un array y almacenar los resultados, como hemos hecho antes con variables sencillas.

```
var miarray = [64, 32];
miarray[1] = miarray[1] + 10;
alert("El valor actual es " + miarray[1]); // "El valor actual es 42"
```

Listado 6-31: Trabajando con los valores del array

Los arrays trabajan exactamente igual que otras variables, con la excepción de que tenemos que mencionar el índice cada vez que queremos usarlos. Con la instrucción **miarray[1] = miarray[1] + 10** le decimos al intérprete de JavaScript que lea el valor actual de **miarray** en el índice 1 (32), le sume 10, y almacene el resultado en el mismo array e índice; por lo que al final el valor de **miarray[1]** es 42.

Los arrays pueden incluir cualquier tipo de valores, por lo que es posible declarar arrays de arrays. Estos tipos de arrays se denominan *arrays multidimensionales*.

```
var miarray = [[2, 45, 31], [5, 10], [81, 12]];
```

Listado 6-32: Definiendo arrays multidimensionales

El ejemplo del Listado 6-32 crea un array de arrays de números enteros. Para acceder a estos valores, tenemos que declarar los índices de cada nivel entre corchetes, uno después del otro. El siguiente ejemplo devuelve el primer valor (índice 0) del segundo array (índice 1). La instrucción busca el array en el índice 1 y luego busca por el número en el índice 0.

```
var miarray = [[2, 45, 31], [5, 10], [81, 12]];
alert(miarray[1][0]);  // 5
```

Listado 6-33: Accediendo a los valores en arrays multidimensionales

Si queremos eliminar uno de los valores, podemos declararlo como **undefined** o **null**, como hemos hecho anteriormente, o declararlo como un array vacío asignando corchetes sin valores en su interior.

```
var miarray = [[2, 45, 31], [5, 10], [81, 12]];
miarray[1] = []
alert(miarray[1][0]);  // undefined
```

Listado 6-34: Asignando un array vacío como el valor de otro array

Ahora, el valor mostrado en la ventana emergente es **undefined**, porque no hay ningún elemento en las posición **[1][0]**. Por supuesto, esto también se puede usar para vaciar cualquier tipo de array.

```
var miarray = [2, 45, 31];
miarray = []
alert(miarray[1]);  // undefined
```

Listado 6-35: Asignando un array vacío a una variable

Lo básico: al igual que las cadenas de caracteres, los arrays también se declaran como objetos en JavaScript y, por lo tanto, incluyen métodos para realizar operaciones en sus valores. Estudiaremos objetos, los objetos **array**, y cómo implementar sus métodos más adelante en este capítulo.

Condicionales y bucles

Hasta este punto hemos escrito instrucciones en secuencia, una debajo de la otra. En este tipo de programas, el sistema ejecuta cada instrucción una sola vez y en el orden en el que se presentan. Comienza con la primera y sigue hasta llegar al final de la lista. El propósito de condicionales y bucles es el de romper esta secuencia. Los condicionales nos permiten ejecutar una o más instrucciones solo cuando se cumple una determinada condición, y los bucles nos permiten ejecutar un bloque de código (un grupo de instrucciones) una y otra vez hasta que se satisface una condición. JavaScript ofrece un total de cuatro instrucciones para procesar código de acuerdo a condiciones determinadas por el programador: **if**, **switch**, **for** y **while**.

La manera más simple de comprobar una condición es con la instrucción **if**. Esta instrucción analiza una expresión y procesa un grupo de instrucciones si la condición establecida por esa expresión es verdadera. La instrucción requiere la palabra clave **if** seguida de la condición entre paréntesis y las instrucciones que queremos ejecutar si la condición es verdadera entre llaves.

```
var mivariable = 9;
if (mivariable < 10) {
  alert("El número es menor que 10");
}
```

Listado 6-36: Comprobando una condición con **if**

En el código del Listado 6-36, el valor 9 se asigna a **mivariable**, y luego, usando **if** comparamos la variable con el número **10**. Si el valor de la variable es menor que **10**, la función **alert()** muestra un mensaje en la pantalla.

El operador usado para comparar el valor de la variable con el número 10 se llama *operador de comparación*. Los siguientes son los operadores de comparación disponibles en JavaScript.

- == comprueba si el valor de la izquierda es igual al de la derecha.
- != comprueba si el valor de la izquierda es diferente al de la derecha.
- > comprueba si el valor de la izquierda es mayor que el de la derecha.
- < comprueba si el valor de la izquierda es menor que el de la derecha.
- >= comprueba si el valor de la izquierda es mayor o igual que el de la derecha.
- <= comprueba si el valor de la izquierda es menor o igual que el de la derecha.

Después de evaluar una condición, esta devuelve un valor lógico verdadero o falso. Esto nos permite trabajar con condiciones como si fueran valores y combinarlas para crear condiciones más complejas. JavaScript ofrece los siguientes operadores lógicos con este propósito.

- ! (negación) permuta el estado de la condición. Si la condición es verdadera, devuelve falso, y viceversa.
- && (y) comprueba dos condiciones y devuelve verdadero si ambas son verdaderas.
- || (o) comprueba dos condiciones y devuelve verdadero si una o ambas son verdaderas.

El operador lógico ! invierte el estado de la condición. Si la condición se evalúa como verdadera, el estado final será falso, y las instrucciones entre llaves no se ejecutarán.

```
var mivariable = 9;
if (!(mivariable < 10)) {
  alert("El número es menor que 10");
}
```

***Listado 6-37:** Invirtiendo el resultado de la condición*

El código del Listado 6-37 no muestra ningún mensaje en la pantalla. El valor 9 es aún menor que 10, pero debido a que alteramos la condición con el operador !, el resultado final es falso, y la función `alert()` no se ejecuta.

Para que el operador trabaje sobre el estado de la condición y no sobre los valores que estamos comparando, debemos encerrar la condición entre paréntesis. Debido a los paréntesis, la condición se evalúa en primer lugar y luego el estado que devuelve se invierte con el operador !.

Los operadores && (y) y || (o) trabajan de un modo diferente. Estos operadores calculan el resultado final basándose en los resultados de las condiciones involucradas. El operador && (y) devuelve verdadero solo si las condiciones a ambos lados devuelven verdadero, y el operador || (o) devuelve verdadero si una o ambas condiciones devuelven verdadero. Por ejemplo, el siguiente código ejecuta la función `alert()` solo cuando la edad es menor de 21 y el valor de la variable `inteligente` es igual a "SI" (debido a que usamos el operador &&, ambas condiciones tienen que ser verdaderas para que la condición general sea verdadera).

```
var inteligente = "SI";
var edad = 19;
if (edad < 21 && inteligente == "SI") {
   alert("Juan está autorizado");
}
```

Listado 6-38: Comprobando múltiples condiciones con operadores lógicos

Si asumimos que nuestro ejemplo solo considera dos valores para la variable **inteligente**, "SI" y "NO", podemos convertirla en una variable booleana. Debido a que los valores booleanos son valores lógicos, no necesitamos compararlos con nada. El siguiente código simplifica el ejemplo anterior usando una variable booleana.

```
var inteligente = true;
var edad = 19;
if (edad < 21 && inteligente) {
   alert("Juan está autorizado");
}
```

Listado 6-39: Usando valores booleanos como condiciones

JavaScript es bastante flexible en cuanto a los valores que podemos usar para establecer condiciones. El lenguaje es capaz de determinar una condición basándose en los valores de cualquier variable. Por ejemplo, una variable con un número entero devolverá falso si el valor es 0 o verdadero si el valor es diferente de 0.

```
var edad = 0;
if (edad) {
   alert("Juan está autorizado");
}
```

Listado 6-40: Usando números enteros como condiciones

El código de la instrucción **if** del Listado 6-40 no se ejecuta porque el valor de la variable **edad** es 0 y, por lo tanto, el estado de la condición se considera falso. Si almacenamos un valor diferente dentro de esta variable, la condición será verdadera y el mensaje se mostrará en la pantalla.

Las variables con cadenas de caracteres vacías también devuelven falso. El siguiente ejemplo comprueba si se ha asignado una cadena de caracteres a una variable y muestra su valor solo si la cadena no está vacía.

```
var nombre = "Juan";
if (nombre) {
   alert(nombre + " está autorizado");
}
```

Listado 6-41: Usando cadenas de caracteres como condiciones

Hágalo usted mismo: reemplace el código en su archivo micodigo.js con el código del Listado 6-41 y abra el documento del Listado 6-5 en su navegador. La ventana emergente muestra el mensaje "Juan está autorizado". Asigne una cadena vacía a la variable **nombre**. La instrucción **if** ahora considera falsa la condición y no se muestra ningún mensaje en pantalla.

A veces debemos ejecutar instrucciones para cada estado de la condición (verdadero o falso). JavaScript incluye la instrucción **if else** para ayudarnos en estas situaciones. Las instrucciones se presentan en dos bloques de código delimitados por llaves. El bloque precedido por **if** se ejecuta cuando la condición es verdadera y el bloque precedido por **else** se ejecuta en caso contrario.

```
var mivariable = 21;
if (mivariable < 10) {
  alert("El número es menor que 10");
} else {
  alert("El numero es igual o mayor que 10");
}
```

Listado 6-42: Comprobando dos condiciones con `if else`

En este ejemplo, el código considera dos condiciones: cuando el número es menor que **10** y cuando el número es igual o mayor que **10**. Si lo que necesitamos es comprobar múltiples condiciones, en lugar de las instrucciones **if else** podemos usar la instrucción **switch**. Esta instrucción evalúa una expresión (generalmente una variable), compara el resultado con múltiples valores y ejecuta las instrucciones correspondientes al valor que coincide con la expresión. La sintaxis incluye la palabra clave **switch** seguida de la expresión entre paréntesis. Los posibles valores se listan usando la palabra clave **case**, tal como muestra el siguiente ejemplo.

```
var mivariable = 8;
switch(mivariable) {
  case 5:
    alert("El número es cinco");
    break;
  case 8:
    alert("El número es ocho");
    break;
  case 10:
    alert("El número es diez");
    break;
  default:
    alert("El número es " + mivariable);
}
```

Listado 6-43: Comprobando un valor con la instrucción `switch`

En el ejemplo del Listado 6-43, la instrucción **switch** evalúa la variable **mivariable** y luego compara su valor con el valor de cada caso. Si el valor es **5**, por ejemplo, el control se transfiere al primer **case**, y la función **alert()** muestra el texto "El número es cinco" en la pantalla. Si el

primer **case** no coincide con el valor de la variable, se evalúa el siguiente caso, y así sucesivamente. Si ningún caso coincide con el valor, se ejecutan las instrucciones en el caso **default**.

En JavaScript, una vez que se encuentra una coincidencia, las instrucciones en ese caso se ejecutan junto con las instrucciones de los casos siguientes. Este es el comportamiento por defecto, pero normalmente no lo que nuestro código necesita. Por esta razón, JavaScript incluye la instrucción **break**. Para evitar que el sistema ejecute las instrucciones de cada caso después de que se encuentra una coincidencia, tenemos que incluir la instrucción **break** al final de cada caso.

Las instrucciones **switch** e **if** son útiles pero realizan una tarea sencilla: evalúan una expresión, ejecutan un bloque de instrucciones de acuerdo al resultado y al final devuelven el control al código principal. En ciertas situaciones esto no es suficiente. A veces tenemos que ejecutar las instrucciones varias veces para la misma condición o evaluar la condición nuevamente cada vez que se termina un proceso. Para estas situaciones, contamos con dos instrucciones: **for** y **while**.

La instrucción **for** ejecuta el código entre llaves mientras la condición es verdadera. Usa la sintaxis **for(inicialización; condición; incremento)**. El primer parámetro establece los valores iniciales del bucle, el segundo parámetro es la condición que queremos comprobar y el último parámetro es una instrucción que determina cómo van a evolucionar los valores iniciales en cada ciclo.

```
var total = 0;
for (var f = 0; f < 5; f++) {
  total += 10;
}
alert("El total es: " + total);  // "El total es: 50"
```

Listado 6-44: Creando un bucle con la instrucción **for**

En el código del Listado 6-44 declaramos una variable llamada **f** para controlar el bucle y asignamos el número **0** como su valor inicial. La condición en este ejemplo comprueba si el valor de la variable **f** es menor que 5. En caso de ser verdadera, se ejecuta el código entre llaves. Después de esto, el intérprete ejecuta el último parámetro de la instrucción **for**, el cual suma 1 al valor actual de **f** (**f++**), y luego comprueba la condición nuevamente (en cada ciclo **f** se incrementa en 1). Si la condición es aún verdadera, las instrucciones se ejecutan una vez más. Este proceso continúa hasta que **f** alcanza el valor **5**, lo cual vuelve falsa la condición (5 no es menor que 5) y el bucle se interrumpe.

Dentro del bucle **for** del Listado 6-44, sumamos el valor 10 al valor actual de la variable **total**. Los bucles se usan frecuentemente de esta manera para hacer evolucionar el valor de una variable de acuerdo a resultados anteriores. Por ejemplo, podemos usar el bucle **for** para sumar todos los valores de un array.

```
var total = 0;
var lista = [23, 109, 2, 9];
for (var f = 0; f < 4; f++) {
  total += lista[f];
}
alert("El total es: " + total);  // "El total es: 143"
```

Listado 6-45: Iterando sobre los valores de un array

Para leer todos los valores de un array, tenemos que crear un bucle que va desde el índice del valor inicial a un valor que coincide con el índice del último valor del array. En este caso, el array `lista` contiene cuatro elementos y, por lo tanto, los índices correspondientes van de 0 a 3. El bucle lee el valor en el índice 0, lo suma al valor actual de la variable `total` y luego avanza hacia el siguiente valor en el array hasta que el valor de `f` es igual a 4 (no existe un valor en el índice 4). Al final, todos los valores del array se suman a la variable `total` y el resultado se muestra en pantalla.

Lo básico: en el ejemplo del Listado 6-45, hemos podido configurar el bucle porque conocemos el número de valores dentro del array, pero esto no es siempre posible. A veces no contamos con esta información durante el desarrollo de la aplicación, ya sea porque el array se crea cuando la página se carga o porque los valores los introduce el usuario. Para trabajar con arrays dinámicos, JavaScript ofrece la propiedad `length`. Esta propiedad devuelve la cantidad de valores dentro de un array. Estudiaremos esta propiedad y los objetos `Array` más adelante en este capítulo.

La instrucción `for` es útil cuando podemos determinar ciertos requisitos, como el valor inicial del bucle o el modo en que evolucionarán esos valores en cada ciclo. Cuando esta información es poco clara, podemos utilizar la instrucción `while`. La instrucción `while` solo requiere la declaración de la condición entre paréntesis y el código a ser ejecutado entre llaves. El bucle se ejecuta constantemente hasta que la condición es falsa.

```
var contador = 0;
while(contador < 100) {
  contador++;
}
alert("El valor es: " + contador);  // "El valor es: 100"
```

Listado 6-46: Usando la instrucción `while`

El ejemplo del Listado 6-46 es sencillo. La instrucción entre llaves se ejecuta mientras el valor de la variable `contador` es menor que 100. Esto significa que el bucle se ejecutará 100 veces (cuando el valor de `contador` es 99, la instrucción se ejecuta una vez más y, por lo tanto, el valor final de la variable es 100).

Si la primera vez que la condición se evalúa devuelve un valor falso (por ejemplo, cuando el valor inicial de `contador` ya es mayor de 99), el código entre llaves nunca se ejecuta. Si queremos que las instrucciones se ejecuten al menos una vez, sin importar cuál sea el resultado de la condición, podemos usar una implementación diferente del bucle `while` llamada `do while`. La instrucción `do while` ejecuta las instrucciones entre llaves y luego comprueba la condición, lo cual garantiza que las instrucciones se ejecutarán al menos una vez. La sintaxis es similar, solo tenemos que preceder las llaves con la palabra clave `do` y declarar la palabra clave `while` con la condición al final.

```
var contador = 150;
do {
  contador++;
} while(contador < 100);
alert("El valor es: " + contador);  // "El valor es: 151"
```

Listado 6-47: Usando la instrucción `do while`

En el ejemplo del Listado 6-47, el valor inicial de la variable **contador** es mayor de 99, pero debido a que usamos el bucle **do while**, la instrucción entre llaves se ejecuta una vez y, por lo tanto, el valor final de **contador** es 151 (150 + 1 = 151).

Instrucciones de transferencia de control

Los bucles a veces se deben interrumpir. JavaScript ofrece múltiples instrucciones para detener la ejecución de bucles y condicionales. Las siguientes son las que más se usan.

continue—Esta instrucción interrumpe el ciclo actual y avanza hacia el siguiente. El sistema ignora el resto de instrucciones del bucle después de que se ejecuta esta instrucción.

break—Esta instrucción interrumpe el bucle. Todas las instrucciones restantes y los ciclos pendientes se ignoran después de que se ejecuta esta instrucción.

La instrucción **continue** se aplica cuando no queremos ejecutar el resto de las instrucciones entre llaves, pero queremos seguir ejecutando el bucle.

```
var lista = [2, 4, 6, 8];

var total = 0;
for (var f = 0; f < 4; f++) {
  var numero = lista[f];
  if (numero == 6) {
    continue;
  }
  total += numero;
}
alert("El total es: " + total);  // "El total es: 14"
```

Listado 6-48: Saltando hacia el siguiente ciclo del bucle

La instrucción **if** dentro del bucle **for** del Listado 6-48 compara el valor de **numero** con el valor 6. Si el valor del array que devuelve la primera instrucción del bucle es 6, se ejecuta la instrucción **continue**, la última instrucción del bucle se ignora, y el bucle avanza hacia el siguiente valor en el array **lista**. En consecuencia, todos los valores del array se suman a la variable **total** excepto el número 6.

A diferencia de **continue**, la instrucción **break** interrumpe el bucle completamente, delegando el control a la instrucción declarada después de bucle.

```
var lista = [2, 4, 6, 8];

var total = 0;
for (var f = 0; f < 4; f++) {
  var numero = lista[f];
  if (numero == 6) {
    break;
  }
  total += numero;
}
alert("El total es: " + total);  // "El total es: 6"
```

Listado 6-49: Interrumpiendo el bucle

Nuevamente, la instrucción `if` del Listado 6-49 compara el valor de `numero` con el valor 6, pero esta vez ejecuta la instrucción `break` cuando los valores coinciden. Si el número del array que devuelve la primera instrucción es 6, la instrucción `break` se ejecuta y el bucle termina, sin importar cuántos valores quedaban por leer en el array. En consecuencia, solo los valores ubicados antes del número 6 se suman al valor de la variable `total`.

6.2 Funciones

Las funciones son bloques de código identificados con un nombre. La diferencia entre las funciones y los bloques de código usados en los bucles y los condicionales estudiados anteriormente es que no hay que satisfacer ninguna condición; las instrucciones dentro de una función se ejecutan cada vez que se llama a la función. Las funciones se llaman (ejecutadas) escribiendo el nombre seguido de paréntesis. Esta llamada se puede realizar desde cualquier parte del código y cada vez que sea necesario, lo cual rompe completamente el procesamiento secuencial del programa. Una vez que una función es llamada, la ejecución del programa continúa con las instrucciones dentro de la función (sin importar dónde se localiza en el código) y solo devuelve a la sección del código que ha llamado la función cuando la ejecución de la misma ha finalizado.

Declarando funciones

Las funciones se declaran usando la palabra clave `function`, el nombre seguido de paréntesis, y el código entre llaves. Para llamar a la función (ejecutarla), tenemos que declarar su nombre con un par de paréntesis al final, como mostramos a continuación.

```
function mostrarMensaje() {
  alert("Soy una función");
}
mostrarMensaje();
```

Listado 6-50: Declarando funciones

Las funciones se deben primero declarar y luego ejecutar. El código del Listado 6-50 declara una función llamada `mostrarMensaje()` y luego la llama una vez. Al igual que con las variables, el intérprete de JavaScript lee la función, almacena su contenido en memoria, y asigna una referencia al nombre de la función. Cuando llamamos a la función por su nombre, el intérprete comprueba la referencia y lee la función en memoria. Esto nos permite llamar a la función todas las veces que sea necesario, como los hacemos en el siguiente ejemplo.

```
var total = 5;
function calcularValores(){
  total = total * 2;
}
for(var f = 0; f < 10; f++){
  calcularValores();
}
alert("El total es: " + total);  // "El total es: 5120"
```

Listado 6-51: Procesando datos con funciones

El ejemplo del Listado 6-51 combina diferentes instrucciones ya estudiadas. Primero declara una variable y le asigna el valor 5. Luego, se declara una función llamada `calcularValores()` (pero no se ejecuta). A continuación, una instrucción `for` se usa para crear un bucle que será ejecutado mientras el valor de la variable `f` sea menor que 10. La instrucción dentro del bucle llama a la función `calcularValores()`, por lo que la función se ejecuta en cada ciclo. Cada vez que la función se ejecuta, el valor actual de `total` se multiplica por 2, duplicando su valor en cada ocasión.

Ámbito

En JavaScript, las instrucciones que se encuentran fuera de una función se considera que están en el ámbito global. Este es el espacio en el que escribimos las instrucciones hasta que se define una función u otra clase de estructura de datos. Las variables definidas en el ámbito global tienen un alcance global y, por lo tanto, se pueden usar desde cualquier parte del código, pero las declaradas dentro de las funciones tienen un alcance local, lo que significa que solo se pueden usar dentro de la función en la que se han declarado. Esta es otra ventaja de las funciones; son lugares especiales en el código donde podemos almacenar información a la que no se podrá acceder desde otras partes del código. Esta segregación nos ayuda a evitar generar duplicados que pueden conducir a errores, como sobrescribir el valor de una variable cuando el valor anterior aún era requerido por la aplicación.

El siguiente ejemplo ilustra cómo se definen los diferentes ámbitos y qué debemos esperar cuando accedemos desde un ámbito a variables que se han definido en un ámbito diferente.

```
var variableGlobal = 5;
function mifuncion(){
  var variableLocal = "El valor es ";
  alert(variableLocal + variableGlobal);  // "El valor es 5"
}
mifuncion();
alert(variableLocal);
```

Listado 6-52: Declarando variables globales y locales

El código del Listado 6-52 declara una función llamada `mifuncion()` y dos variables, una en el ámbito global llamada `variableGlobal` y otra dentro de la función llamada `variableLocal`. Cuando se ejecuta la función, el código concatena las variables `variableLocal` y `variableGlobal`, y muestra la cadena de caracteres obtenida en la pantalla. Debido a que `variableGlobal` es una variable global, es accesible dentro de la función y, por lo tanto, su valor se agrega a la cadena de caracteres, pero cuando intentamos mostrar el valor de `variableLocal` fuera de la función, nos devuelve un error (la ventana emergente no se muestra). Esto se debe a que `variableLocal` se ha definido dentro de la función y, por lo tanto, no es accesible desde el ámbito global.

Lo básico: los navegadores informan de los errores producidos por el código JavaScript en una consola oculta. Si necesita ver los errores porque genera su código, tiene que abrir esta consola desde las opciones del menú del navegador (Herramientas, en Google Chrome). Al final de este capítulo estudiaremos más estas consolas y cómo controlar errores.

Debido a que las variables declaradas en diferentes ámbitos se consideran diferentes variables, dos variables con el mismo nombre, una en el ámbito global y otra en el ámbito local (dentro de una función), se considerarán dos variables distintas (se les asigna un espacio de memoria diferente).

```
var mivariable = 5;
function mifuncion(){
  var mivariable = "Esta es una variable local";
  alert(mivariable);
}
mifuncion();
alert(mivariable);
```

Listado 6-53: Declarando dos variables con el mismo nombre

En el código del Listado 6-53, declaramos dos variables llamadas **mivariable**, una en el ámbito global y la otra dentro de la función **mifuncion()**. También incluimos dos funciones **alert()** en el código, una dentro de la función **mifuncion()** y otra en el ámbito global al final del código. Ambas muestran el contenido de la variable **mivariable**, pero referencian distintas variables. La variable dentro de la función está referenciando un espacio en memoria que contiene la cadena de caracteres "Esta es una variable local", mientras que la variable en el ámbito global está referenciando un espacio en memoria que contiene el valor 5.

Lo básico: las variables globales también se pueden crear desde las funciones. Omitir la palabra clave **var** cuando declaramos una variable dentro de una función es suficiente para configurar esa variable como global.

Las variables globales son útiles cuando varias funciones deben compartir valores, pero debido a que son accesibles desde cualquier parte del código, siempre existe la posibilidad de sobrescribir sus valores por accidente desde otras instrucciones, o incluso desde otros códigos (todos los códigos JavaScript incluidos en el documento comparten el mismo ámbito global). Por consiguiente, usar variables globales desde una función no es una buena idea. Una mejor alternativa es enviar valores a las funciones cuando son llamadas.

Para poder recibir un valor, la función debe incluir un nombre entre los paréntesis con el que representar el valor. Estos nombres se denominan *parámetros*. Cuando la función se ejecuta, estos parámetros se convierten en variables que podemos leer desde dentro de la función y así acceder a los valores recibidos.

```
function mifuncion(valor) {
  alert(valor);
}
mifuncion(5);
```

Listado 6-54: Enviando un valor a una función

En el ejemplo del Listado 6-54, dejamos de usar variables globales. El valor a procesar se envía a la función cuando es llamada y esta lo recibe a través de su parámetro. Cuando se llama a la función, el valor entre los paréntesis de la llamada (5) se asigna a la variable **value** creada para recibirlo, y esta variable se lee dentro de la función para mostrar el valor en pantalla.

Lo básico: los nombres declarados entre los paréntesis de la función para recibir valores se llaman *parámetros*. Por otro lado, los valores especificados en la llamada se denominan *atributos*. En estos términos, podemos decir que la llamada a la función tiene atributos que se envían a la función y se reciben mediante sus parámetros.

La ventaja de usar funciones es que podemos ejecutar sus instrucciones una y otra vez, y como podemos enviar diferentes valores en cada llamada, el resultado obtenido en cada una de ellas será diferente. El siguiente ejemplo llama a la función **mifuncion()** dos veces, pero en cada oportunidad envía un valor diferente para ser procesado.

```
function mifuncion(valor) {
  alert(valor);
}
mifuncion(5);
mifuncion(25);
```

Listado 6-55: Llamando a la misma función con diferentes valores

El intérprete ejecuta la primera llamada con el valor 5 y cuando la ejecución de la función finaliza, se llama nuevamente con el valor 25. En consecuencia, se abren dos ventanas emergentes, una con el valor 5 y la otra con el valor 25.

En este y los ejemplos anteriores, enviamos números enteros a la función, pero también podemos enviar el valor actual de una variable.

```
var contador = 100;
function mifuncion(valor) {
  alert(valor);
}
mifuncion(contador);
```

Listado 6-56: Enviando el valor de una variable a una función

Esta vez incluimos la variable **contador** en la llamada en lugar de un número. El intérprete lee esta variable y envía su valor a la función. El resto del proceso es el mismo: la función recibe el valor, lo asigna a la variable **valor** y lo muestra en pantalla.

Las funciones también pueden recibir múltiples valores. Todo lo que tenemos que hacer es declarar los valores y parámetros separados por comas.

```
var contador = 100;
var items = 5;

function mifuncion(valor1, valor2) {
  var total = valor1 + valor2;
  alert(total);  // 105
}
mifuncion(contador, items);
```

Listado 6-57: Enviando múltiples valores a la función

En el ejemplo del Listado 6-57, sumamos los valores recibidos por la función y mostramos el resultado en pantalla, pero a veces este resultado se requiere fuera de la función. Para enviar valores desde la función al ámbito global, JavaScript incluye la instrucción **return**. Esta instrucción determina el valor a devolver al código que ha llamado a la función.

Si queremos procesar el valor que devuelve la función, tenemos que asignar dicha función a una variable. El intérprete primero ejecuta la función y luego asigna el valor que devuelve la función a la variable, tal como muestra el siguiente ejemplo.

```
var contador = 100;
var items = 5;
function mifuncion(valor1, valor2) {
  var total = valor1 + valor2;
  return total;
}
var resultado = mifuncion(contador, items);
alert(resultado);
```

Listado 6-58: Devolviendo valores desde funciones

El código del Listado 6-58 define la misma función **mifuncion()** usada anteriormente, pero esta vez el valor producido por la función no se muestra en la pantalla, sino que se devuelve con la instrucción **return**. De regreso al ámbito global, el valor devuelto por la función se asigna a la variable **resultado** y el contenido de esta variable se muestra en pantalla.

La variable **miresultado** se ha declarado al comienzo del código, pero no le hemos asignamos ningún valor. Esto es aceptable, e incluso recomendado. Es una buena práctica declarar todas las variables con las que vamos a trabajar al comienzo para evitar confusión y poder identificar cada una de ellas más adelante desde otras partes del código.

La instrucción **return** finaliza la ejecución de la función. Cualquier instrucción declarada después de que se devuelve un valor no se ejecutará. Por esta razón, la instrucción **return** normalmente se declara al final de la función, pero esto no es obligatorio. Podemos devolver un valor desde cualquier parte del código si tenemos condiciones que satisfacer. Por ejemplo, la siguiente función devuelve el resultado de la suma de dos valores si el total es mayor que 100, o devuelve el valor 0 en caso contrario.

```
var contador = 100;
var items = 5;
function mifuncion(valor1, valor2) {
  var total = valor1 + valor2;
  if (total > 100) {
    return total;
  } else {
    return 0;
  }
}
var resultado = mifuncion(contador, items);
alert(resultado);
```

Listado 6-59: Devolviendo diferentes valores desde una función

Funciones anónimas

Otra manera de declarar una función es usando funciones anónimas. Las funciones anónimas son funciones sin un nombre o identificador. Debido a esto, se pueden pasar a otras funciones o asignar a variables. Cuando una función anónima se asigna a una variable, el nombre de la variable es el que usamos para llamar a la función, tal como hacemos en el siguiente ejemplo.

```
var mifuncion = function(valor) {
  valor = valor * 2;
  return valor;
};
var total = 2;
for (var f = 0; f < 10; f++) {
  total = mifuncion(total);
}
alert("El total es " + total);  // "El total es 2048"
```

Listado 6-60: Declarando funciones anónimas

En el ejemplo del Listado 6-60, declaramos una función anónima que recibe un valor, lo multiplica por 2 y devuelve el resultado. Debido a que la función se asigna a una variable, podemos usar el nombre de la variable para llamarla, por lo que después de que se define la función, creamos un bucle **for** que llama a la función **mifuncion()** varias veces con el valor actual de la variable **total**. La instrucción del bucle asigna el valor que devuelve la función de vuelta a la variable **total**, duplicando su valor en cada ciclo.

Las funciones anónimas se pueden ejecutar al instante agregando paréntesis al final de su declaración. Esto es útil cuando queremos asignar el resultado de una operación compleja a una variable. La función procesa la operación y devuelve el resultado. En este caso, no es la función lo que se asigna a la variable, sino el valor que devuelve la misma.

```
var mivalor = function(valor) {
  valor = valor * 2;
  return valor;
}(35);
alert("El valor es " + mivalor);  // "El valor es 70"
```

Listado 6-61: Ejecutando funciones anónimas

La función del Listado 6-61 se define y ejecuta tan pronto como el intérprete procesa la instrucción. La función recibe el valor 35, lo multiplica por 2 y devuelve el resultado, que se asigna a la variable **mivalor**.

Lo básico: las funciones anónimas son extremadamente útiles en JavaScript porque nos permiten definir complicados patrones de programación, necesarios para construir aplicaciones profesionales. Estudiaremos ejemplos prácticos del uso de estos tipos de funciones y algunos patrones disponibles en JavaScript en próximos capítulos.

Funciones estándar

Además de las funciones que podemos crear nosotros mismos, también tenemos acceso a funciones predefinidas por JavaScript. Estas funciones realizan procesos que simplifican tareas complejas. Las siguientes son las que más se usan.

isNaN(valor)—Esta función devuelve `true` (verdadero) si el valor entre paréntesis no es un número.

parseInt(valor)—Esta función convierte una cadena de caracteres con un número en un número entero que podemos procesar en operaciones aritméticas.

parseFloat(valor)—Esta función convierte una cadena de caracteres con un número en un número decimal que podemos procesar en operaciones aritméticas.

encodeURIComponent(valor)—Esta función codifica una cadena de caracteres. Se utiliza para codificar los caracteres de un texto que puede crear problemas cuando se inserta en una URL.

decodeURIComponent(valor)—Esta función decodifica una cadena de caracteres.

Las funciones estándar son funciones globales que podemos llamar desde cualquier parte del código; solo tenemos que llamarlas como lo hacemos con cualquier otra función con los valores que queremos procesar entre paréntesis.

```
var mivalor = "Hola";
if (isNaN(mivalor)) {
  alert(mivalor + " no es un número");
} else {
  alert(mivalor + " es un número");
}
```


Listado 6-62: Comprobando si un valor es un número o no

La función `isNaN()` devuelve un valor booleano, por lo que podemos usarla para establecer una condición. El intérprete primero llama a la función y luego ejecuta los bloques de código definidos por las instrucciones `if else` dependiendo del resultado. En este caso, el valor de la variable es una cadena de caracteres, por lo que la función `isNaN()` devuelve el valor `true` y el mensaje "Hola no es un número" se muestra en pantalla.

La función `isNaN()` devuelve el valor `false` no solo cuando la variable contiene un número, sino además cuando contiene una cadena de caracteres con un número. Esto significa que no podemos usar el valor en una operación aritmética porque podría ser una cadena de caracteres y el proceso no produciría el resultado esperado. Para asegurarnos de que el valor se puede incluir en una operación, tenemos que convertirlo en un valor numérico. Para este propósito, JavaScript ofrece dos funciones: `parseInt()` para números enteros y `parseFloat()` para números decimales.

```
var mivalor = "32";
if (isNaN(mivalor)) {
  alert(mivalor + " no es un número");
```

```
} else {
  var numero = parseInt(mivalor);
  numero = numero * 10;
  alert("El número es: " + numero);  // "El número es 320"
}
```

Listado 6-63: Convirtiendo una cadena de caracteres en un número

El código del Listado 6-63 comprueba si el valor de la variable **mivalor** es un número o no, como hemos hecho antes, pero esta vez el valor se convierte en un valor numérico si se encuentra un número. Después de que el valor se extrae de la cadena de caracteres, lo usamos para realizar una multiplicación y mostrar el resultado en pantalla.

Otra función estándar útil es **encodeURIComponent()**. Con esta función podemos preparar una cadena de caracteres para ser incluida en una URL. El problema con las URL es que le otorgan un significado especial a algunos caracteres, como ? o &, como hemos visto en capítulos anteriores (ver Figura 2-43). Debido a que los usuarios no conocen estas restricciones, tenemos que codificar las cadenas de caracteres antes de incluirlas en una URL cada vez que las introduce el usuario o provienen de una fuente que no es fiable.

```
var nombre = "Juan Perez";
var codificado = encodeURIComponent(nombre);
var miURL = "http://www.ejemplo.com/contacto.html?nombre=" +
codificado;
alert(miURL);
```

Listado 6-64: Codificando una cadena de caracteres para incluirla en una URL

El código del Listado 6-64 agrega el valor de la variable **nombre** a una URL. En este ejemplo, asumimos que el valor de la variable lo ha definido el usuario y, por lo tanto, lo codificamos con la función **encodeURIComponent()** para asegurarnos de que la URL final es válida. La función analiza la cadena de caracteres y reemplaza cada carácter conflictivo por un número hexadecimal precedido por el carácter %. En este caso, el único carácter que requiere codificación es el espacio entre el nombre y el apellido. La URL resultante es http://www.ejemplo.com/contacto.html?nombre=Juan%20Perez.

6.3 Objetos

Los objetos son estructuras de información capaces de contener variables (llamadas *propiedades*), así como funciones (llamadas *métodos*). Debido a que los objetos almacenan valores junto con funciones, son como programas independientes que se comunican entre sí para realizar tareas comunes.

La idea detrás de los objetos en programación es la de simular el rol de los objetos en la vida real. Un objeto real tiene propiedades y realiza acciones. Por ejemplo, una persona tiene un nombre y una dirección postal, pero también puede caminar y hablar. Las características y la funcionalidad son parte de la persona y es la persona la que define cómo va a caminar y lo que va a decir. Organizando nuestro código de esta manera, podemos crear unidades de procesamiento independientes capaces de realizar tareas y que cuentan con toda la información que necesitan para hacerlo. Por ejemplo, podemos crear un objeto que controla

un botón, muestra su título, y realiza una tarea cuando se pulsa el botón. Debido a que toda la información necesaria para presentar y controlar el botón se almacena dentro del objeto, el resto del código no necesita saber cómo hacerlo. Siempre y cuando conozcamos los métodos provistos por el objeto y los valores devueltos, el código dentro del objeto se puede actualizar o reemplazar por completo sin afectar el resto del programa.

Poder crear unidades de procesamiento independientes, duplicar esas unidades tantas veces como sea necesario, y modificar sus valores para adaptarlos a las circunstancias actuales, son las principales ventajas introducidas por los objetos y la razón por la que la programación orientada a objetos es el paradigma de programación disponible más popular. JavaScript se creó en torno al concepto de objetos y, por lo tanto, entender objetos es necesario para entender el lenguaje y sus posibilidades.

Declarando objetos

Existen diferentes maneras de declarar objetos en JavaScript, pero la más sencilla es usar notación literal. El objeto se declara como cualquier otra variable usando la palabra clave **var**, y las propiedades y métodos que definen el objeto se declaran entre llaves usando dos puntos después del nombre y una coma para separar cada declaración.

```
var miobjeto = {
  nombre: "Juan",
  edad: 30
};
```

Listado 6-65: Creando objetos

En el ejemplo del Listado 6-65 declaramos el objeto **miobjeto** con dos propiedades: **nombre** y **edad**. El valor de la propiedad **nombre** es "Juan" y el valor de la propiedad **edad** es 30.

A diferencia de las variables, no podemos acceder a los valores de las propiedades de un objeto usando solo sus nombres; también tenemos que especificar el nombre del objeto al que pertenecen usando notación de puntos o corchetes.

```
var miobjeto = {
  nombre: "Juan",
  edad: 30
};
var mensaje = "Mi nombre es " + miobjeto.nombre + "\r\n";
mensaje += "Tengo " + miobjeto["edad"] + " años";
alert(mensaje);
```

Listado 6-66: Accediendo propiedades

En el Listado 6-66, implementamos ambas técnicas para acceder a los valores de las propiedades del objeto y crear el mensaje que vamos a mostrar en la pantalla. El uso de cualquiera de estas técnicas es irrelevante, excepto en algunas circunstancias. Por ejemplo, cuando necesitamos acceder a la propiedad a través del valor de una variable, tenemos que usar corchetes.

```
var nombrePropiedad = "nombre";
var miobjeto = {
  nombre: "Juan",
  edad: 30
};
alert(miobjeto[nombrePropiedad]);  // "Juan"
```

Listado 6-67: Accediendo propiedades usando variables

En el Listado 6-67, no podríamos haber accedido a la propiedad usando notación de puntos (**miobjeto.nombrePropiedad**) porque el intérprete habría intentado acceder a una propiedad llamada **nombrePropiedad** que no existe. Usando corchetes, primero la variable se resuelve y luego se accede al objeto con su valor ("nombre") en lugar de su nombre.

También es necesario acceder a una propiedad usando corchetes cuando su nombre se considera no válido para una variable (incluye caracteres no válidos, como un espacio, o comienza con un número). En el siguiente ejemplo, el objeto incluye una propiedad cuyo nombre se ha declarado con una cadena de caracteres. Está permitido declarar nombres de propiedades con cadena de caracteres, pero como este nombre contiene un espacio, el código **miobjeto.mi edad** produciría un error, por lo que tenemos que usar corchetes para acceder a esta propiedad.

```
var mivariable = "nombre";
var miobjeto = {
  nombre: "Juan",
  'mi edad': 30
};
alert(miobjeto['mi edad']);  // 30
```

Listado 6-68: Accediendo propiedades con nombres no válidos

Además de leer los valores de las propiedades, también podemos asignar nuevas propiedades al objeto o modificarlas usando notación de puntos. En el siguiente ejemplo, modificamos el valor de la propiedad **nombre** y agregamos una nueva propiedad llamada **trabajo**.

```
var miobjeto = {
  nombre: "Juan",
  edad: 30
};
miobjeto.nombre = "Martín";
miobjeto.trabajo = "Programador";
alert(miobjeto.nombre + " " + miobjeto.edad + " " + miobjeto.trabajo);
```

Listado 6-69: Actualizando valores y agregando nuevas propiedades a un objeto

Los objetos también pueden contener otros objetos. En el siguiente ejemplo, asignamos un objeto a la propiedad de otro objeto.

```
var miobjeto = {
  nombre: "Juan",
  edad: 30,
  motocicleta: {
    modelo: "Susuki",
    fecha: 1981
  }
};
alert(miobjeto.nombre + " tiene una " + miobjeto.motocicleta.modelo);
```

Listado 6-70: Creando objetos dentro de objetos

El objeto **miobjeto** en el código del Listado 6-70 incluye una propiedad llamada **motocicleta** cuyo valor es otro objeto con las propiedades **modelo** y **fecha**. Si queremos acceder a estas propiedades, tenemos que indicar el nombre del objeto al que pertenecen (**motocicleta**) y el nombre del objeto al que ese objeto pertenece (**miobjeto**). Los nombres se concatenan con notación de puntos en el orden en el que se han incluido en la jerarquía. Por ejemplo, la propiedad **modelo** está dentro de la propiedad **motocicleta** que a la vez está dentro de la propiedad **miobjeto**. Por lo tanto, si queremos leer el valor de la propiedad **modelo**, tenemos que escribir **miobjeto.motocicleta.modelo**.

Métodos

Como mencionamos anteriormente, los objetos también pueden incluir funciones. Las funciones dentro de los objetos se llaman *métodos*. Los métodos tienen la misma sintaxis que las propiedades: requieren dos puntos después del nombre y una coma para separar cada declaración, pero en lugar de valores, debemos asignarles funciones anónimas.

```
var miobjeto = {
  nombre: "Juan",
  edad: 30,
  mostrardatos: function() {
    var mensaje = "Nombre: " + miobjeto.nombre + "\r\n";
    mensaje += "Edad: " + miobjeto.edad;
    alert(mensaje);
  },
  cambiarnombre: function(nombrenuevo) {
    miobjeto.nombre = nombrenuevo;
  }
};
miobjeto.mostrardatos();  // "Nombre: Juan Edad: 30"
miobjeto.cambiarnombre("José");
miobjeto.mostrardatos();  // "Nombre: José Edad: 30"
```

Listado 6-71: Declarando y ejecutando métodos

En este ejemplo, agregamos dos métodos al objeto: **mostrardatos()** y **cambiarnombre()**. El método **mostrardatos()** muestra una ventana emergente con los valores de las propiedades **nombre** y **edad**, y el método **cambiarnombre()** asigna el valor recibido por su parámetro a la

propiedad **nombre**. Estos son dos métodos independientes que trabajan sobre las mismas propiedades, uno lee sus valores y el otro les asigna nuevos. Para ejecutar los métodos, usamos notación de puntos y paréntesis después del nombre, como hacemos con funciones.

Al igual que las funciones, los métodos también pueden devolver valores. En el siguiente ejemplo, modificamos el método **cambiarnombre()** para devolver el nombre anterior después de que se reemplaza por el nuevo.

```
var miobjeto = {
  nombre: "Juan",
  edad: 30,
  mostrardatos: function() {
    var mensaje = "Nombre: " + miobjeto.nombre + "\r\n";
    mensaje += "Edad: " + miobjeto.edad;
    alert(mensaje);
  },
  cambiarnombre: function(nombrenuevo) {
    var nombreviejo = miobjeto.nombre;
    miobjeto.nombre = nombrenuevo;
    return nombreviejo;
  }
};
var anterior = miobjeto.cambiarnombre("José");
alert("El nombre anterior era: " + anterior);  // "Juan"
```

Listado 6-72: Devolviendo valores desde métodos

El nuevo método **cambiarnombre()** almacena el valor actual de la propiedad **nombre** en una variable temporal llamada **nombreviejo** para poder devolver el valor anterior después de que el nuevo se asigna a la propiedad.

La palabra clave this

En los últimos ejemplos, mencionamos el nombre del objeto cada vez que queríamos modificar sus propiedades desde los métodos. Aunque esta técnica funciona, no es una práctica recomendada. Debido a que el nombre del objeto queda determinado por el nombre de la variable al que se asigna el objeto, el mismo se puede modificar sin advertirlo. Además, como veremos más adelante, JavaScript nos permite crear múltiples objetos desde la misma definición o crear nuevos objetos a partir de otros, lo cual produce diferentes objetos que comparten la misma definición. Para asegurarnos de que siempre referenciamos al objeto con el que estamos trabajando, JavaScript incluye la palabra clave **this**. Esta palabra clave se usa en lugar del nombre del objeto para referenciar el objeto al que la instrucción pertenece. El siguiente ejemplo reproduce el código anterior, pero esta vez usamos la palabra clave **this** en lugar del nombre del objeto para referenciar sus propiedades. El resultado es el mismo que antes.

```
var miobjeto = {
  nombre: "Juan",
  edad: 30,
  mostrardatos: function() {
    var mensaje = "Nombre: " + this.nombre + "\r\n";
    mensaje += "Edad: " + this.edad;
```

```
    alert(mensaje);
  },
  cambiarnombre: function(nombrenuevo) {
    var nombreviejo = this.nombre;
    this.nombre = nombrenuevo;
    return nombreviejo;
  }
};
var anterior = miobjeto.cambiarnombre("José");
alert("El nombre anterior era: " + anterior);  // "Juan"
```

Listado 6-73: Referenciando las propiedades del objeto con la palabra clave `this`

IMPORTANTE: cada vez que queremos acceder a propiedades y métodos desde el interior de un objeto, debemos usar la palabra clave **this** para referenciar el objeto, pero si intentamos hacer lo mismo desde fuera del objeto, en su lugar estaremos referenciando el objeto global de JavaScript. La palabra clave **this** referencia el objeto en el que la instrucción se está ejecutando. Esta es la razón por la que en el código del Listado 6-73 solo usamos la palabra clave **this** dentro de los métodos del objeto **miobjeto**, pero las instrucciones en el ámbito global siguen usando el nombre del objeto.

Constructores

Usando notación literal podemos crear objetos individuales, pero si queremos crear copias de estos objetos con las mismas propiedades y métodos, tenemos que usar constructores. Un constructor es una función anónima que define un nuevo objeto y lo devuelve, creando copias del objeto (también llamadas *instancias*), cada una con sus propias propiedades, métodos y valores.

```
var constructor = function() {
  var obj = {
    nombre: "Juan",
    edad: 30,
    mostrarnombre: function() {
      alert(this.nombre);
    },
    cambiarnombre: function(nombrenuevo) {
      this.nombre = nombrenuevo;
    }
  };
  return obj;
};
var empleado = constructor();
empleado.mostrarnombre();  // "Juan"
```

Listado 6-74: Usando un constructor para crear un objeto

En el ejemplo del Listado 6-74 se asigna una función anónima a la variable **constructor**. Dentro de la función, se crea un objeto y se devuelve mediante la instrucción **return**. Finalmente, el objeto que devuelve la función se almacena en la variable **empleado** y se ejecuta su método **mostrarnombre()**.

Usando constructores, podemos crear nuevos objetos de forma dinámica. Por ejemplo, podemos configurar valores iniciales para las propiedades enviando los valores a la función cuando se construye el objeto.

```
var constructor = function(nombreinicial) {
  var obj = {
    nombre: nombreinicial,
    edad: 30,
    mostrarnombre: function() {
      alert(this.nombre);
    },
    cambiarnombre: function(nombrenuevo) {
      this.nombre = nombrenuevo;
    }
  };
  return obj;
};
var empleado = constructor("Juan");
empleado.mostrarnombre();  // "Juan"
```

Listado 6-75: Enviando valores iniciales al constructor

El propósito de un constructor es el de funcionar como una fábrica de objetos. El siguiente ejemplo ilustra cómo crear múltiples objetos con el mismo constructor.

```
var constructor = function(nombreinicial) {
  var obj = {
    nombre: nombreinicial,
    edad: 30,
    mostrarnombre: function() {
      alert(this.nombre);
    },

    cambiarnombre: function(nombrenuevo) {
      this.nombre = nombrenuevo;
    }
  };
  return obj;
};
var empleado1 = constructor("Juan");
var empleado2 = constructor("Roberto");
var empleado3 = constructor("Arturo");
alert(empleado1.nombre + ", " + empleado2.nombre + ", " +
empleado3.nombre);
```

Listado 6-76: Usando constructores para crear múltiples objetos

Aunque los objetos creados desde un constructor comparten las mismas propiedades y métodos, se almacenan en diferentes espacios de memoria y, por lo tanto, manipulan valores diferentes. Cada vez que se llama la función `constructor`, se crea un nuevo objeto y podemos almacenar diferentes valores en cada uno de ellos. En el ejemplo del Listado 6-76, hemos creado tres objetos: `empleado1`, `empleado2`, y `empleado3`, y se ha asignado los

valores "Juan", "Roberto", y "Arturo" a sus propiedades **nombre**. En consecuencia, cuando leemos la propiedad **nombre** de cualquiera de estos objetos, obtenemos diferentes valores dependiendo del objeto al que pertenece la propiedad (la función **alert()** al final del código muestra el mensaje "Juan, Roberto, Arturo").

Una ventaja de usar constructores para crear objetos es la posibilidad de definir propiedades y métodos privados. Todo los objetos que hemos creado hasta el momento contienen propiedades y métodos públicos, lo cual significa que se puede acceder a sus contenidos y modificarlos desde cualquier parte del código. Para evitar esto último y hacer que las propiedades y métodos solo sean accesibles mediante el objeto que los ha creado, tenemos que volverlos privados usando una técnica llamada *closure*.

Como hemos explicado anteriormente, se puede acceder a las variables creadas en el ámbito global desde cualquier lugar del código, mientras que a las variables creadas dentro de funciones solo se puede acceder desde las funciones en las que se han creado. Lo que no mencionamos es que las funciones y, por lo tanto, los métodos mantienen un enlace que las conecta al ámbito en el que se han creado y quedan conectadas a las variables declaradas en ese ámbito. Cuando devolvemos un objeto desde un constructor, sus métodos aún pueden acceder a las variables de la función, incluso cuando ya no se encuentran en el mismo ámbito, y por ello estas variables se vuelven accesibles solo para el objeto.

```
var constructor = function() {
  var nombre = "Juan";
  var edad = 30;
  var obj = {
    mostrarnombre: function() {
        alert(nombre);
    },
    cambiarnombre: function(nombrenuevo) {
      nombre = nombrenuevo;
    }
  };
  return obj;
};
var empleado = constructor();
empleado.mostrarnombre();  // "Juan"
```

***Listado 6-77:** Definiendo propiedades privadas*

El código del Listado 6-77 es exactamente el mismo que hemos definido en el ejemplo anterior excepto que en lugar de declarar **nombre** y **edad** como propiedades del objeto, las declaramos como variables de la función que está devolviendo el objeto. El objeto devuelto recordará estas variables, pero el mismo será el único que tendrá acceso a ellas. No existe forma de modificar los valores de esas variables desde otras instrucciones en el código que no sea por medio de los métodos que devuelve la función (en este caso, **mostrarnombre()** y **cambiarnombre()**). Esta es la razón por la que el procedimiento se denomina *closure* (clausura). La función se cierra y no se puede acceder a su ámbito, pero mantenemos un elemento conectado a ella (un objeto en nuestro ejemplo).

Los métodos en este ejemplo acceden a las variables sin usar la palabra clave **this**. Esto se debe a que los valores ahora se almacenan en variables definidas por la función y no en propiedades definidas por el objeto.

Lo básico: cada nuevo objeto creado por un constructor se almacena en un espacio diferente en la memoria y, por lo tanto, tiene la misma estructura y sus propias variables privadas y valores, pero también podemos asignar el mismo objeto a distintas variables. Si quiere asegurarse de que una variable no está referenciando al mismo objeto, puede comparar las variables con los operadores especiales `===` y `!==`. JavaScript también incluye el método `is()` dentro de un objeto global llamado `Object` que podemos usar para comprobar si dos variables referencian el mismo objeto (por ejemplo, `Object.is(objeto1, objeto2)`).

El operador new

Con la notación literal y los constructores tenemos todo lo que necesitamos para crear objetos individuales o múltiples objetos basados en una misma definición, pero para ser coherente con otros lenguajes de programación orientada a objetos, JavaScript ofrece una tercera alternativa. Se trata de una clase especial de constructor que trabaja con un operador llamado **new** (nuevo). El objeto se define mediante una función y luego se llama con el operador new para crear un objeto a partir de esa definición.

```
function MiObjeto() {
  this.nombre = "Juan";
  this.edad = 30;
  this.mostrarnombre = function(){
    alert(this.nombre);
  };
  this.cambiarnombre = function(nombrenuevo){
    this.nombre = nombrenuevo;
  };
}
var empleado = new MiObjeto();
empleado.mostrarnombre();  // "Juan"
```

Listado 6-78: Creando objetos con el operador new

Estos tipos de constructores requieren que las propiedades y los métodos de los objetos sean identificados mediante la palabra clave `this`, pero excepto por este requisito, la definición de estos constructores y los que hemos estudiado anteriormente son iguales. También podemos proveer valores iniciales, como en el siguiente ejemplo.

```
function MiObjeto(nombreinicial, edadinicial){
  this.nombre = nombreinicial;
  this.edad = edadinicial;
  this.mostrarnombre = function(){
    alert(this.nombre);
  };
  this.cambiarnombre = function(nombrenuevo){
    this.nombre = nombrenuevo;
  };
}
var empleado = new MiObjeto("Roberto", 55);
```

```
empleado.mostrarnombre();  // "Roberto"
```

Listado 6-79: Definiendo valores iniciales para el objeto

Herencia

Una característica importante de los objetos es que podemos crearlos desde otros objetos. Cuando los objetos se crean a partir de otros objetos, pueden heredar sus propiedades y métodos, y también agregar los suyos propios. La herencia en JavaScript (cómo los objetos obtienen las mismas propiedades y métodos de otros objetos) se logra a través de prototipos. Un objeto no hereda las propiedades y los métodos de otro objeto directamente; lo hace desde el prototipo del objeto. Trabajando con prototipos puede resultar muy confuso, pero JavaScript incluye el método **279áximo()** para simplificar nuestro trabajo. Este método es parte de un objeto global predefinido por JavaScript llamado **Object**. El método utiliza un objeto que ya existe como prototipo de uno nuevo, de modo que podemos crear objetos a partir de otros objetos sin preocuparnos de cómo se comparten entre ellos las propiedades y los métodos.

```
var miobjeto = {
  nombre: "Juan",
  edad: 30,
  mostrarnombre: function(){
    alert(this.nombre);
  },
  cambiarnombre: function(nombrenuevo){
    this.nombre = nombrenuevo;
  }
};
var empleado = Object.create(miobjeto);
empleado.cambiarnombre('Roberto');
empleado.mostrarnombre();  // "Roberto"
miobjeto.mostrarnombre();  // "Juan"
```

Listado 6-80: Creando objetos a partir de otros objetos

El código del Listado 6-80 crea el objeto **miobjeto** usando notación literal y luego llama al método **279áximo()** para crear un nuevo objeto basado en el objeto **miobjeto**. El método **áximo()** solo requiere el nombre del objeto que se va usar como prototipo del nuevo objeto, y devuelve este nuevo objeto que podemos asignar a una variable para su uso posterior. En este ejemplo, el nuevo objeto se crea con **Object.create()** y luego se asigna a la variable **empleado**. Una vez que tenemos el nuevo objeto, podemos actualizar sus valores. Usando el método **cambiarnombre()**, cambiamos el nombre de **empleado** a "Roberto" y luego mostramos el valor de la propiedad **nombre** de cada objeto en la pantalla.

Este código crea dos objetos independientes, **miobjeto** y **empleado**, con sus propias propiedades, métodos y valores, pero conectados a través de la cadena de prototipos. El nuevo objeto **empleado** no es solo una copia del original, es un objeto que mantiene un enlace con el prototipo de **miobjeto**. Cuando introducimos cambios a este prototipo, los objetos siguientes en la cadena heredan estos cambios.

```
var miobjeto = {
```

```
  nombre: "Juan",
  edad: 30,
  mostrarnombre: function(){
    alert(this.nombre);
  },

cambiarnombre: function(nombrenuevo){
    this.nombre = nombrenuevo;
  }
};
var empleado = Object.create(miobjeto);
empleado.edad = 24;

miobjeto.mostraredad = function(){
  alert(this.edad);
};
empleado.mostraredad();  // 24
```

Listado 6-81: Agregando un nuevo método al prototipo

En el Listado 6-81, se agrega un método llamado **mostraredad()** al objeto prototipo (**miobjeto**). Debido a la cadena de prototipos, este nuevo método es accesible también desde las otras instancias. Cuando llamamos al método **mostraredad()** de **empleado** al final del código, el intérprete busca el método primero en **empleado** y continúa buscando hacia arriba en la cadena de prototipos hasta que lo encuentra en el objeto **miobjeto**. Cuando finalmente se encuentra el método, muestra el valor 24 en la pantalla. Esto se debe a que, a pesar de que el método **mostraredad()** se ha definido dentro de **miobjeto**, la palabra clave **this** en este método apunta al objeto con el que estamos trabajando (**empleado**). Debido a la cadena de prototipos, se puede invocar al método **mostraredad()** desde **empleado**, y debido a la palabra clave **this**, el valor de la propiedad **edad** que se muestra en la pantalla es el que se ha asignado a **empleado**.

El método **280áximo()** es tan sencillo como poderoso: toma un objeto y lo convierte en el prototipo de uno nuevo. Esto nos permite construir una cadena de objetos donde cada uno hereda las propiedades y los métodos de su predecesor.

```
var miobjeto = {
  nombre: "Juan",
  edad: 30,
  mostrarnombre: function(){
    alert(this.nombre);
  },
  cambiarnombre: function(nombrenuevo){
    this.nombre = nombrenuevo;
  }
};
var empleado1 = Object.create(miobjeto);
var empleado2 = Object.create(empleado1);
var empleado3 = Object.create(empleado2);

empleado2.mostraredad = function(){
  alert(this.edad);
};
```

```
empleado3.edad = 24;
empleado3.mostraredad();  // 24
```

Listado 6-82: Probando la cadena de prototipos

En el Listado 6-82, la existencia de la cadena de prototipos se demuestra agregando el método `mostraredad()` a `empleado2`. Ahora, `empleado2` y `empleado3` (y también cualquier otro objeto creado posteriormente a partir de estos dos objetos) tienen acceso a este método, pero debido a que la herencia funciona hacia abajo de la cadena y no hacia arriba, el método no está disponible para el objeto `empleado1`.

IMPORTANTE: si se acostumbra a trabajar con el método `281áximo()`, no necesitará acceder y modificar los prototipos de los objetos directamente. De hecho, el método `281áximo()` debería ser la forma estándar de trabajar en JavaScript, creando objetos sin tener que lidiar con la complejidad de su sistema de prototipos. Sin embargo, los prototipos son la esencia de este lenguaje y en algunas circunstancias no podemos evitarlos. Para obtener más información sobre los prototipos y cómo trabajar con objetos, visite nuestro sitio web y siga los enlaces de este capítulo.

6.4 Objetos estándar

Los objetos son como envoltorios de código y JavaScript se aprovecha de esta característica extensamente. De hecho, casi todo en JavaScript es un objeto. Por ejemplo, los números y las cadenas de caracteres que asignamos a las variables se convierten automáticamente en objetos por el intérprete JavaScript. Cada vez que asignamos un nuevo valor a una variable, en realidad estamos asignando un objeto que contiene ese valor.

Debido a que los valores que almacenamos en variables son de tipos diferentes, existen diferentes tipos de objetos disponibles para representarlos. Por ejemplo, si es valor es una cadena de caracteres, el objeto creado para almacenarlo es del tipo `String`. Cuando asignamos un texto a una variable, JavaScript crea un objeto `String`, almacena la cadena de caracteres en el objeto y asigna ese objeto a la variable. Si queremos, podemos crear estos objetos directamente usando sus constructores. Existen diferentes constructores disponibles dependiendo del tipo de valor que queremos almacenar. Los siguientes son los más usados.

Number(valor)—Este constructor crea objetos para almacenar valores numéricos. Acepta números y también cadenas de caracteres con números. Si el valor especificado por el atributo no se puede convertir en un número, el constructor devuelve el valor `NaN` (No es un Número).

String(valor)—Este constructor crea objetos para almacenar cadenas de caracteres. Acepta una cadena de caracteres o cualquier valor que pueda convertirse en una cadena de caracteres, incluidos números.

Boolean(valor)—Este constructor crea objetos para almacenar valores booleanos. Acepta los valores `true` y `false`. Si el valor se omite o es igual a 0, `NaN`, `null`, `undefined`, o una cadena de caracteres vacía, el valor almacenado en el objeto es `false`, en caso contrario es `true`.

Array(valores)—Este constructor crea objetos para almacenar arrays. Si se proveen múltiples valores, el constructor crea un array con esos valores, pero si solo se provee un valor, y ese valor es un número entero, el constructor crea un array con la cantidad de elementos que indica el valor del atributo y almacena el valor **undefined** en cada índice.

Estos constructores trabajan con el operador **new** para crear nuevos objetos. El siguiente ejemplo almacena un número.

```
var valor = new Number(5);
alert(valor);  // 5
```

Listado 6-83: Creando números con un constructor

La ventaja de usar constructores es que pueden procesar diferentes tipos de valores. Por ejemplo, podemos obtener un número a partir de una cadena de caracteres que contiene un número.

```
var valor = new Number("5");
alert(valor);  // 5
```

Listado 6-84: Creando números a partir de cadenas de caracteres

La cadena de caracteres que provee al constructor **Number()** en el Listado 6-84 se convierte en un valor numérico y se almacena en un objeto **Number**. Debido a que JavaScript se encarga de convertir estos objetos en valores primitivos y viceversa, podemos realizar operaciones aritméticas sobre el valor almacenado en el objeto, como lo hacemos con cualquier otro valor numérico.

```
var valor = new Number("5");
var total = valor * 35;
alert(total);  // 175
```

Listado 6-85: Realizando operaciones aritméticas con objetos

El constructor **Array()** se comporta de un modo diferente. Si especificamos varios valores, el array se crea como si lo hubiéramos declarado con corchetes.

```
var lista = new Array(12, 35, 57, 8);
alert(lista);  // 12,35,57,8
```

Listado 6-86: Creando un array con un constructor

Por otro lado, si especificamos un único valor y ese valor es un número entero, el constructor lo utiliza para determinar el tamaño del array, crea un array con esa cantidad de elementos y asigna el valor **undefined** a cada uno de ellos.

```
var lista = new Array(2);
alert(lista[0] + " - " + lista[1]);  // undefined - undefined
```

Listado 6-87: Creando un array vacío con un constructor

Lo básico: el método que utilicemos para asignar un valor depende de nosotros. Podemos asignar los valores directamente como hemos hecho en ejemplos previos o usar estos constructores. El resultado es el mismo. JavaScript es capaz de determinar la diferencia entre un valor y un objeto, pero generalmente esto resulta irrelevante, y la mayoría del tiempo los dos tipos de valores se comportan de la misma manera.

Objetos String

Convertir valores en objetos permite al lenguaje ofrecer la funcionalidad necesaria para manipular esos valores. Las siguientes son las propiedades y los métodos más usados de los objetos `String`.

Length—Esta propiedad devuelve un entero que representa la cantidad de caracteres en la cadena.

toLowerCase()—Este método convierte los caracteres a letras minúsculas.

toUpperCase()—Este método convierte los caracteres a letras mayúsculas.

Trim()—Este método elimina espacios en blanco a ambos lados de la cadena de caracteres. JavaScript también incluye los métodos `trimLeft()` y `trimRight()` para limpiar la cadena de caracteres en un lado específico (izquierda o derecha).

Substr(comienzo, extensión**)**—Este método devuelve una nueva cadena de caracteres con caracteres extraídos de la cadena original. El atributo **comienzo** indica la posición del primer carácter a leer, y el atributo **extensión** determina cuántos caracteres queremos incluir. Si no se especifica la extensión, el método devuelve todos los caracteres hasta el final de la cadena.

Substring(comienzo, final**)**—Este método devuelve una nueva cadena de caracteres con caracteres extraídos de la cadena original. Los atributos **comienzo** y **final** son números enteros que determinan las posiciones del primer y último carácter a incluir.

283áxim(separador, limite**)**—Este método divide la cadena de caracteres y devuelve un array con todas las partes. El atributo **separador** indica el carácter en el que se va a cortar la cadena y el atributo **límite** es un número entero que determina el número máximo de partes. Si no especificamos un límite, la cadena se cortará cada vez que se encuentre el separador.

startsWith(valor**)**—Este método devuelve `true` si el texto especificado por el atributo **valor** se encuentra al comienzo de la cadena de caracteres.

endsWith(valor**)**—Este método devuelve `true` si el texto especificado por el atributo **valor** se encuentra al final de la cadena de caracteres.

Includes(buscar, posición)—Este método busca el valor del atributo **buscar** dentro de la cadena y devuelve **true** o **false** de acuerdo con el resultado. El atributo **buscar** es el texto que queremos buscar, y el atributo **posición** determina el índice en el que queremos comenzar la búsqueda. Si el atributo **posición** no se especifica, la búsqueda comienza desde el inicio de la cadena.

indexOf(valor, posición)—Este método devuelve el índice en el que el texto especificado por el atributo **valor** se encuentra por primera vez. El atributo **posición** determina el índice en el que queremos comenzar la búsqueda. Si el atributo **posición** no se especifica, la búsqueda comienza desde el inicio de la cadena. El método devuelve el valor -1 si ninguna coincidencia es encontrada.

lastIndexOf(valor, posición)—Este método devuelve el índice en el que se encuentra por primera vez el texto especificado por el atributo **valor**. A diferencia de **indexOf()**, este método realiza una búsqueda hacia atrás, desde el final de la cadena. El atributo **posición** determina el índice en el que queremos comenzar la búsqueda. Si no se especifica el atributo **posición**, la búsqueda comienza desde el final de la cadena. El método devuelve el valor -1 si no se encuentra ninguna coincidencia.

Replace(expresión, reemplazo)—Este método reemplaza la parte de la cadena de caracteres que coincide con el valor del atributo **expresión** por el texto especificado por el atributo **reemplazo**. El atributo **expresión** se puede especificar como una cadena de caracteres o como una expresión regular para buscar un texto con un formato particular.

Las cadenas de caracteres se almacenan como arrays de caracteres, de modo que podemos acceder a cada carácter usando corchetes, como lo hacemos con cualquier otro array. JavaScript incluye la propiedad **length** para contar la cantidad de caracteres en la cadena.

```
var texto = "Hola Mundo";
var mensaje = "El texto tiene " + texto.length + " caracteres";
alert(mensaje);  // "El texto tiene 10 caracteres"
```

Listado 6-88: Contando la cantidad de caracteres en una cadena

Debido a que las cadenas de caracteres se almacenan como arrays, podemos iterar sobre los caracteres con un bucle. En el siguiente ejemplo, agregamos un espacio entre las letras de un texto.

```
var texto = "Hola Mundo";
var mensaje = "";
for (var f = 0; f < texto.length; f++) {
  mensaje += texto[f] + " ";
}
alert(mensaje);  // "H o l a   M u n d o "
```

Listado 6-89: Iterando sobre los caracteres de una cadena

La propiedad **length** devuelve el número de caracteres en la cadena, pero los índices comienzan a contar desde 0, por lo que debemos crear un bucle que vaya desde 0 al valor

anterior de la propiedad **length** para obtener todos los caracteres en la cadena. Usando estos índices, el bucle **for** del ejemplo del Listado 6-89 lee cada carácter usando corchetes y los agrega al valor actual de la variable **mensaje** junto con un espacio. En consecuencia, obtenemos una nueva cadena de caracteres con todos los caracteres de la cadena original separados por un espacio.

Este ejemplo agrega un espacio después de cada carácter en la cadena, lo cual significa que el texto final termina con un espacio en blanco. Los objetos **String** ofrecen el método **trim()** para eliminar estos espacios no deseados.

```
var texto = "Hola Mundo";
var mensaje = "";
for (var f = 0; f < texto.length; f++) {
  mensaje += texto[f] + " ";
}
mensaje = mensaje.trim();
alert(mensaje);  // "H o l a   M u n d o"
```

Listado 6-90: Removiendo espacios

La posibilidad de acceder a cada carácter en una cadena nos permite lograr efectos interesantes. Solo tenemos que detectar la posición del carácter que queremos manipular y realizar los cambios que deseamos. El siguiente ejemplo agrega puntos entre las letras de cada palabra, pero no entre las palabras.

```
var texto = "Hola Mundo";
var mensaje = "";
var anterior = "";

for (var f = 0; f < texto.length; f++) {
  if (mensaje != "") {
    anterior = texto[f - 1];
    if (anterior != " " && texto[f] != " ") {
      mensaje += ".";
    }
  }
  mensaje += texto[f];
}
alert(mensaje);  // "H.o.l.a M.u.n.d.o"
```

Listado 6-91: Procesando una cadena de caracteres

El código del Listado 6-91 comprueba si el carácter que estamos leyendo y el que hemos leído en el ciclo anterior no son espacios en blanco antes de agregar un punto al valor actual de la variable **mensaje**. De este modo, los puntos se insertan solo entre las letras y no al final o comienzo de cada palabra.

Debido a la complejidad que pueden alcanzar algunos de los procesos realizados con cadenas de caracteres, JavaScript incluye varios métodos para simplificar nuestro trabajo. Por ejemplo, podemos reemplazar todos los caracteres en una cadena con letras mayúsculas con solo llamar al método **toUpperCase()**.

```
var texto = "Hola Mundo";
var mensaje = texto.toUpperCase();
alert(mensaje);  // "HOLA MUNDO"
```

Listado 6-92: Convirtiendo los caracteres de una cadena en letras mayúsculas

A veces no necesitamos trabajar con todo el texto, sino solo con algunas palabras o caracteres. Los objetos **String** incluyen los métodos **substr()** y **substring()** para copiar un trozo de texto desde una cadena. El método **substr()** copia el grupo de caracteres que comienza en el índice especificado por el primer atributo. También se puede especificar un segundo atributo para determinar cuántos caracteres queremos incluir desde la posición inicial.

```
var texto = "Hola Mundo";
var palabra = texto.substr(0, 4);
alert(palabra);  // "Hola"
```

Listado 6-93: Copiando un grupo de caracteres

El método **substr()** del Listado 6-93 copia un total de cuatro caracteres comenzando con el carácter en el índice 0. Como resultado, obtenemos la cadena "Hola". Si no especificamos el número de caracteres a incluir, el método devuelve todos los caracteres hasta que llega al final de la cadena.

```
var texto = "Hola Mundo";
var palabra = texto.substr(5);
alert(palabra);  // "Mundo"
```

Listado 6-94: Copiando todos los caracteres desde un índice hasta el final de la cadena

El método **substr()** también puede tomar valores negativos. Cuando se especifica un índice negativo, el método cuenta de atrás hacia adelante. El siguiente código copia los mismos caracteres que el ejemplo anterior.

```
var texto = "Hola Mundo";
var palabra = texto.substr(-5);
alert(palabra);  // "Mundo"
```

Listado 6-95: Referenciando caracteres con índices negativos

El método **substring()** trabaja de una forma diferente al método **substr()**. Este método toma dos valores para determinar los caracteres primero y último que queremos copiar, pero no incluye el último carácter.

```
var texto = "Hola Mundo";
var palabra = texto.substring(5, 7);
alert(palabra);  // "Mu"
```

Listado 6-96: Copiando caracteres entre dos índices

El método **substring()** del Listado 6-96 copia los caracteres desde el índice 5 al 7 de la cadena (el carácter en el último índice no se incluye). Estas son las posiciones en las que se encuentran los caracteres "Mu" y, por lo tanto, este es el valor que obtenemos en respuesta.

Si lo que necesitamos es extraer las palabras de una cadena, podemos usar el método **287áxim()**. Este método corta la cadena en partes más pequeñas y devuelve un array con estos valores. El método requiere un valor con el carácter que queremos usar como separador, por lo que si usamos un espacio, podemos dividir la cadena en palabras.

```
var texto = "Hola Mundo";
var palabras = texto.split(" ");
alert(palabras[0] + " / " + palabras[1]);  // "Hola / Mundo"
```

Listado 6-97: Obteniendo las palabras de una cadena

El método **287áxim()** del Listado 6-97 crea dos cadenas de caracteres con las palabras "Hola" y "Mundo", y devuelve un array con estos valores, que podemos leer como hacemos con los valores de cualquier otro array.

A veces no sabemos dónde se encuentran los caracteres que queremos modificar o si la cadena incluye esos caracteres. Existen varios métodos disponibles en los objetos **String** para hacer una búsqueda. El que usemos depende de lo que queremos lograr. Por ejemplo, los métodos **startsWith()** y **endsWith()** buscan un texto al comienzo o al final de la cadena, y devuelven **true** si se encuentra el texto.

```
var texto = "Hola Mundo";
if (texto.startsWith("Ho")) {
  alert("El texto comienza con 'Ho'");
}
```

Listado 6-98: Buscando un texto al comienzo de la cadena

Debido a que estos métodos devuelven valores booleanos, podemos usarlos para establecer la condición de una instrucción **if**. En el código del Listado 6-98, buscamos el texto "Ho" al comienzo de la variable **texto** y mostramos un mensaje en caso de éxito. En este ejemplo, se encuentra el texto, el método devuelve **true** y, por lo tanto, el mensaje se muestra en pantalla.

Si el texto que estamos buscando puede encontrarse en cualquier parte de la cadena, no solo al comienzo o al final, podemos usar el método **includes()**. Al igual que los métodos anteriores, el método **includes()** busca un texto y devuelve **true** en caso de éxito, pero la búsqueda se realiza en toda la cadena.

```
var texto = "Hola Mundo";
if (texto.includes("l")) {
  alert("El texto incluye la letra L");
}
```

Listado 6-99: Buscando un texto dentro de otro texto

Hasta el momento, hemos comprobado si uno o más caracteres se encuentran dentro de una cadena, pero a veces necesitamos saber dónde se han encontrado esos caracteres. Existen

dos métodos para este propósito: **indexOf()** y **lastIndexOf()**. Ambos métodos devuelven el índice donde se encuentra la primera coincidencia, pero el método **indexOf()** comienza la búsqueda desde el inicio de la cadena y el método **lastIndexOf()** lo hace desde el final.

```
var texto = "Hola Mundo";
var 288áximo = texto.indexOf("Mundo");
alert("La palabra comienza en el índice " + 288áximo);  // 5
```

Listado 6-100: Encontrando la ubicación de un texto dentro de una cadena de caracteres

Una vez que conocemos la posición de los caracteres que estamos buscando, podríamos crear un pequeño código que reemplace esos caracteres por otros diferentes, pero esto no es necesario. Los objetos **String** incluyen un método llamado **replace()** con este propósito. Este es un método complejo que puede tomar múltiples valores y trabajar no solo con cadenas de caracteres, sino con expresiones regulares, pero también podemos usarlo con textos breves o palabras. El siguiente ejemplo reemplaza la palabra "Mundo" por la palabra "Planeta".

```
var texto = "Hola Mundo";
var textonuevo = texto.replace("Mundo", "Planeta");
alert(textonuevo);  // "Hola Planeta"
```

Listado 6-101: Reemplazando textos en una cadena de caracteres

Objetos Array

Como ya mencionamos, en JavaScript los arrays también son objetos e incluyen propiedades y métodos para manipular sus valores. Los siguientes son los más usados.

Length—Esta propiedad devuelve un número entero que representa la cantidad de valores en el array.

Push(valores)—Este método agrega uno o más valores al final del array y devuelve la nueva extensión del array. También contamos con un método similar llamado **unshift()**, que agrega los valores al comienzo del array.

Pop()—Este método elimina el último valor del array y lo devuelve. También contamos con un método similar llamado **shift()**, que elimina el primer valor del array.

Concat(array)—Este método concatena el array con el array especificado por el atributo y devuelve un nuevo array con el resultado. Los arrays originales no se modifican.

Splice(índice, remover, valores)—Este método agrega o elimina valores de un array y devuelve un nuevo array con los elementos eliminados. El atributo **índice** indica el índice en el que vamos a introducir las modificaciones, el atributo **remover** es el número de valores que queremos eliminar desde el índice, y el atributo **valores** es la lista de valores separados por coma que queremos agregar al array desde el índice. Si solo queremos agregar valores, el atributo **remover** se puede declarar con el valor 0, y si solo queremos eliminar valores, podemos ignorar el último atributo.

Slice(comienzo, final)—Este método copia los valores en las posiciones indicadas por los atributos dentro de un nuevo array. Los atributos **comienzo** y **final** indican los índices del primer y último valor a copiar. El último valor no se incluye en el nuevo array.

indexOf(valor, posición)—Este método devuelve el índice en el que se encuentra por primera vez el valor especificado por el atributo **valor**. El atributo **posición** determina el índice en el que queremos comenzar la búsqueda. Si el atributo **posición** no se especifica, la búsqueda comienza desde el inicio del array. El método devuelve el valor **-1** si no se encuentra ninguna coincidencia.

lastIndexOf(valor, posición)—Este método devuelve el índice en el que el valor especificado por el atributo **valor** se encuentra por primera vez. A diferencia de `indexOf()`, este método realiza una búsqueda de atrás hacia adelante. El atributo **posición** determina el índice en el que queremos comenzar la búsqueda. Si no se especifica el atributo **posición**, la búsqueda comienza desde el final del array. El método devuelve el valor -1 si no se encuentra ninguna coincidencia.

Filter(función)—Este método envía los valores del array a una función uno por uno y devuelve un nuevo array con todos los valores que aprueba la función. El atributo **función** es una referencia a una función o una función anónima a cargo de validar los valores. La función recibe tres valores: el valor a evaluar, su índice y una referencia al array. Después de procesar el valor, la función debe devolver un valor booleano para indicar si es válido o no.

every(función)—Este método envía los valores del array a una función uno por uno y devuelve `true` cuando la función aprueba todos los valores. El atributo **función** es una referencia a una función o una función anónima a cargo de evaluar los valores. La función recibe tres valores: el valor a evaluar, su índice, y una referencia al array. Después de procesar el valor, la función debe devolver un valor booleano indicando si es válido o no. También contamos con un método similar llamado `some()` que devuelve `true` si la función aprueba al menos uno de los valores.

Join(separador)—Este método crea una cadena de caracteres con todos los valores en el array. El atributo **separador** especifica el carácter o la cadena de caracteres que queremos incluir entre los valores.

Reverse()—Este método invierte el orden de los valores en el array.

Sort(función)—Este método ordena los valores en el array. El atributo **función** es una referencia a una función o una función anónima a cargo de comparar los valores. La función recibe dos valores desde el array y debe devolver un valor booleano indicando su orden. Si no se especifica el atributo, el método ordena los elementos alfabéticamente y en orden ascendente.

Map(función)—Este método envía los valores del array a una función uno por uno y crea un nuevo array con los valores que devuelve la función. El atributo `función` es una referencia a una función o una función anónima a cargo de procesar los valores. La función recibe tres valores: el valor a procesar, su índice y una referencia al array.

Al igual que los objetos `String`, los objetos `Array` ofrecen la propiedad `length` para obtener la cantidad de valores en el array. La implementación es la misma; solo tenemos que leer la propiedad para obtener el valor en respuesta.