hello_

03.3

COLECCIÓN FRONTEND

CSS3 for Everybody

(porque el estilo no debería ser un dolor de cabeza)

Aprende desde cero cómo funciona CSS

Crea diseños flexibles, coloridos y responsivos

Domina propiedades, clases, layouts y

un día descubrí que display: flex; es magia negra… pero útil

@mihifidem

Desarrollador front-end, formador y luchador incansable contra los !important

Diseño web adaptable con CSS

Tabla de contenidos

Tema 03.3—Diseño web adaptable

Introducción

Introducción

El responsive design es un enfoque de diseño web que permite que un sitio web se adapte automáticamente a diferentes tamaños de pantalla y dispositivos. Esto se logra mediante el uso de medios CSS3 como la regla @media y las unidades de medida relativas (porcentajes, vh y vw). Con estas herramientas, los desarrolladores pueden definir diferentes estilos para diferentes tamaños de pantalla y garantizar que su sitio web se vea bien en cualquier dispositivo, desde smartphones hasta escritorios. Esto mejora la experiencia del usuario y aumenta la accesibilidad del sitio web.

Además de las reglas @media y las unidades de medida relativas, existen otros conceptos clave en el diseño responsive que pueden ayudar a crear un sitio web adaptable:

- Uso de plantillas flexibles: las plantillas flexibles permiten que los elementos de una página se ajusten automáticamente a diferentes tamaños de pantalla, manteniendo la estructura y la disposición de los elementos en todo momento.

- Diseño de grillas: las grillas son una serie de líneas horizontales y verticales que se superponen en una página y ayudan a organizar el contenido. En un diseño responsive, las grillas se pueden ajustar dinámicamente para adaptarse a diferentes tamaños de pantalla.

- Imágenes y medios flexibles: las imágenes y otros medios deben ser flexibles y ajustarse automáticamente a diferentes tamaños de pantalla para mantener la calidad visual y evitar problemas de visualización.

- Uso de medios de remplazo: los medios de reemplazo, como la propiedad "background-size: cover" en CSS, permiten que las imágenes se ajusten automáticamente al tamaño de la pantalla sin distorsión.

El diseño responsive es un aspecto importante en el desarrollo de sitios web modernos y garantiza una experiencia de usuario consistente en todos los dispositivos.

Tema 03.3

Diseño web adaptable

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 3.3-1: *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 3.3-2: *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 3.3-3: *Definiendo los estilos para pantallas pequeñas (adaptablecelulares.css)*

Ejercicio 1:cree un nuevo archivo HTML con el documento del Listado 3.3-1. Cree un archivo CSS llamado adaptabletodos.css con la regla del Listado 3.3-2 y un archivo CSS llamado adaptablecelulares.css con la regla del Listado 3.3-3. 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 3.3-4: *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).

Ejercicio 2:reemplace las reglas en su archivo adaptabletodos.css por el código del Listado 3.3-4 y elimine el segundo elemento **<link>** del documento del Listado 3.3-1 (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 específica 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 3.3-5: *Incluyendo múltiples puntos de interrupción*

Cuando se asignan los estilos del Listado 3.3-5 al documento del Listado 3.3-1, 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.

Ejercicio 3:reemplace las reglas del archivo adaptabletodos.css con el código del Listado 3.3-5. 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 omenos, 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 3.3-6: *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 3.3-6, 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 3.3-7: *Creando elementos flexibles con valores en porcentaje*

Las reglas del Listado 3.3-7 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 3.3-1: *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 3.3- 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 3.3-8: *Agregando márgenes y relleno a elementos flexibles*

Las reglas del Listado 3.3-8 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 3.3-2: *Elementos flexibles con relleno y márgenes*

Ejercicio 3:reemplace las reglas del archivo adaptabletodos.css por el código del Listado 3.3-8. Incluya la imagen miimagen.jpg en el directorio del documento. Abra el documento en su navegador. Debería ver algo similar a la Figura 3.3-2.

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 3.3-9: *Incluyendo el relleno y el borde en el tamaño del elemento*

En el Listado 3.3-9 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).

Ejercicio 4:reemplace las reglas del archivo adaptabletodos.css con el código del Listado 3.3-9. Abra el documento en su navegador. Debería ver algo similar a la Figura 3.3-2, 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 dedeclarar 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 3.3-10: *Declarando columnas fijas y flexibles*

Las reglas del Listado 3.3-10 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 3.3-10 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 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 3.3-3: *Columnas flexibles y fijas*

Ejercicio 5:reemplace las reglas en su archivo adaptabletodos.css por el código del Listado 3.3-10. 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 3.3-6) 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 3.3-11: *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.

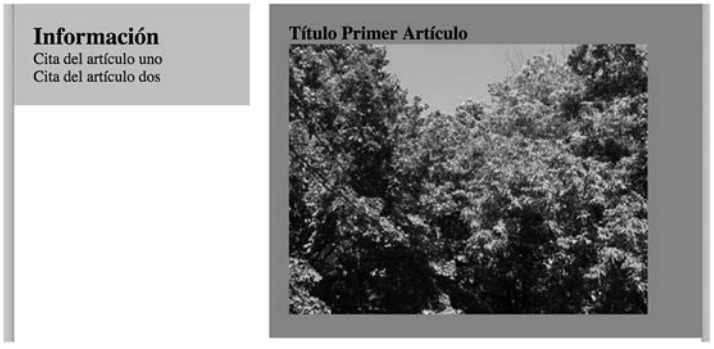

Figura 3.3-4: *Columna fija del lado izquierdo*

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 3.3-12: *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 3.3-13: *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 3.3-5: *Columnas fijas a ambos lados*

Ejercicio 6:cree un nuevo archivo HTML con el documento del Listado 3.3-12 y un archivo CSS llamado misestilos.css con el código del Listado 3.3-13. 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 3.3-5, 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 3.3-6).

```
* {
  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 3.3-14: *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 píxeles 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 3.3-14 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).

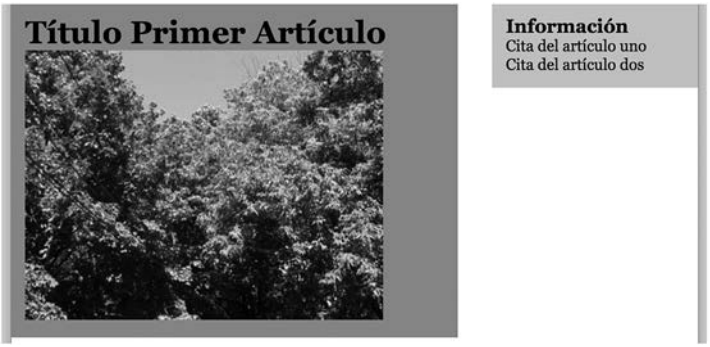

Figura 3.3-6: *Tamaño de letra relativo*

Ejercicio 7:el ejemplo del Listado 3.3-14 asume que estamos trabajando con el documento del Listado 3.3-6. Reemplace las reglas en el archivo adaptabletodos.css creado para este documento con el código del Listado 3.3-14 yabra el documento en su navegador. Debería ver algo similar a la Figura 3.3-6.

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 3.3-15: *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 3.3-7.

Figura 3.3-7: *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 3.3-16: *Declarando tamaños relativos para los elementos con unidades* `rem`

Si aplicamos las reglas del Listado 3.3-16 al ejemplo del Listado 3.3-14, 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 3.3-6, 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 3.3-17: *Adaptando el tamaño de las imágenes*

Ejercicio 8:agregue la regla del Listado 3.3-17 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 3.3-18: *Declarando un tamaño máximo para las imágenes*

La regla del Listado 3.3-18 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.

Ejercicio 9:reemplace la regla del Listado 3.3-17 en su hoja de estilo con la regla del Listado 3.3-18. 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 3.3-8.

Figura 3.3-8: *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 3.3-19: *Seleccionando la imagen con el elemento* `<picture>`

En el documento del Listado 3.3-19 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 3.3-9: *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 3.3-10: *Logo para pantallas pequeñas*

Ejercicio 10:cree un nuevo archivo HTML con el documento del Listado 3.3-19. 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 Figura3.3-9 o la Figura 3.3-10.

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 3.3-20: *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 3.3-20, 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 3.3-21: *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 3.3-11 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).

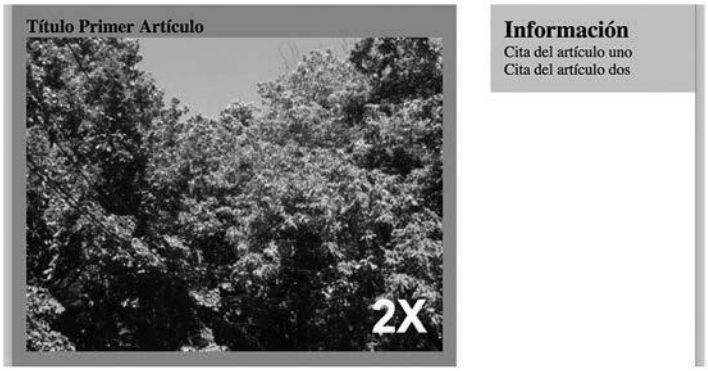

Figura 3.3-11: *Imagen de alta resolución mostrada en una pantalla Retina*

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 3.3-18), 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 3.3-22: *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 3.3-12: *Limitaciones para imágenes de alta resolución*

Ejercicio 11:cree un nuevo archivo HTML con el documento del Listado 3.3-20 y un archivo CSS llamado misestilos.css con el código del Listado 3.3-21. Descargue las imágenes miimagen.jpg y miimagen@2x.jpg desde nuestro sitio web y muévalas al directorio del documento. Abra eldocumento en su navegador. Si su ordenador incluye una pantalla Retina, debería ver la imagen con el texto "2X" en la parte inferior, tal como ilustrala Figura 3.3-11. Actualice la hoja de estilo con la regla del Listado 3.3-22 ycargue 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 3.3-23: *Incluyendo el elemento necesario para mostrar la imagen de fondo*

La cabecera de este documento contiene un elemento 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 3.3-24: *Actualizando la imagen de fondo*

La hoja de estilo del Listado 3.3-24 define el tamaño del elemento igual al tamaño de

la imagen que queremos mostrar y luego asigna la imagen logo.jpg como su imagen de fondo (ver Figura 3.3-9). Si el área de visualización es de 480 píxeles o inferior, la Media Query ajusta el tamaño del elemento para que coincida con el tamaño del logo pequeño y asigna la imagen logoreducido.jpg al fondo del elemento (ver Figura 3.3-10). El efecto que produce este código es exactamente el mismo que el que se logra con el elemento <picture> en el documento del Listado 3.3-19, excepto que esta vez la imagen no la presenta el elemento **<img>**, sino que se asigna al fondo del elemento ** **.

Ejercicio 12:cree un nuevo archivo HTML con el documento del Listado 3.3-23 y un archivo CSS llamado misestilos.css con el código del Listado 3.3-24. Descargue las imágenes logo.jpg y logoreducido.jpg desde nuestro sitioweb 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 3.3-9 o la Figura 3.3-10.

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 3.3-25: *Definiendo elementos flexibles con el modelo de caja tradicional*

Las reglas del Listado 3.3-25 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 3.3-13.

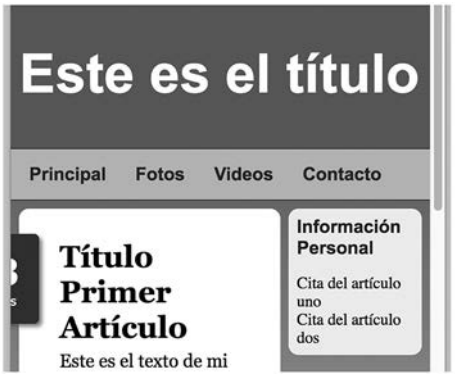

Figura 3.3-13: *Sitio web adaptable solo con elementos flexibles*

Ejercicio 13: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 3.3-25. 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 3.3-13). 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 3.3-26: *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 3.3-25 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 3.3-14).

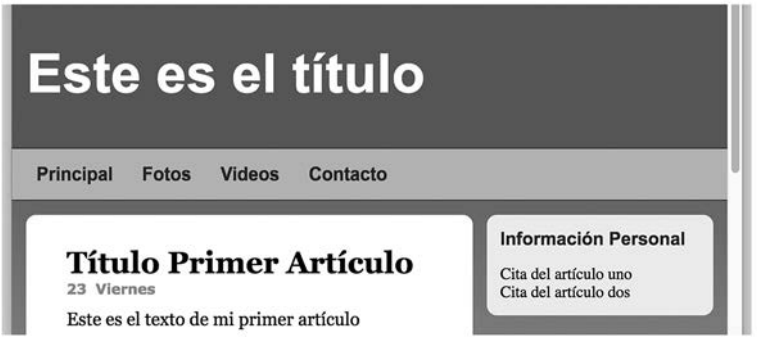

Figura 3.3-14: *La fecha se desplaza a la posición debajo del título con Media Queries*

Ejercicio 14:agregue la Media Query definida en el Listado 3.3-26 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 3.3-27: *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.

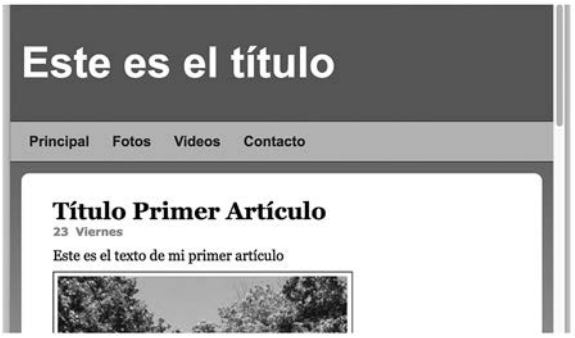

Figura 3.3-15: *Diseño de una columna*

Ejercicio 14:agregue la Media Query definida en el Listado 3.3-27 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 3.3-15.

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 3.3-28: *Creando un menú móvil*

Con las reglas del Listado 3.3-28 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 3.3-16 ilustra cómo afectan estos cambios a algunos de los elementos.

Figura 3.3-16: *Menú móvil*

Ejercicio 15:agregue la Media Query definida en el Listado 3.3-28 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 3.3-29: *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 3.3-30: *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 3.3-31: *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 3.3-17: *Botón del menú*

Ejercicio 16:agregue un elemento **<nav>** identificado con el nombre **menuicono** encima del elemento **<nav>** que ya existe en su documento, como muestra el Listado 3.3-29. Agregue la regla del Listado 3.3-30 en la parte superior de su hoja de estilo. Actualice la Media Query para el punto de interrupción 480px con el código del Listado 3.3-31. 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 3.3-32: *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 3.3-32 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 3.3-18 muestra lo que vemos cuando se pulsa el botón.

Ejercicio 17:agregue el código del Listado 3.3-32 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 3.3-18.

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 Tema 3.2, Listado 3.2-52.

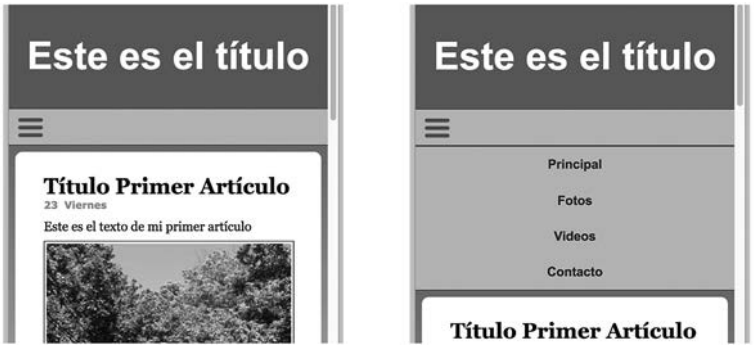

Figura 3.3-18: *Menú mostrado por código JavaScript*

Este ejemplo asume que hemos incluido en el documento el elemento <nav> agregado en el Listado 3.3-29 y el elemento <script> con el código JavaScript introducido en el Listado 3.3-32, 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 3.3-33: *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 3.3-34: *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 3.3-35: *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 3.3-36: *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 3.3-37: *Adaptando el menú y el pie de página*

Con el código del Listado 3.3-37, 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.

Ejercicio 18:cree un nuevo archivo HTML con el documento del Listado4-52 (vea el modelo de caja flexible en el Capítulo 4). Agregue el elemento **<nav>** introducido en el Listado 3.3-29 y el elemento **<script>** del Listado 3.3-33al documento tal como hemos explicado en el ejemplo anterior. Cree un nuevo archivo CSS llamado misestilos.css y copie los códigos de los Listados 3.3-34, 3.3-35,3.3-36 y 3.3-37 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.