hello_

03.2

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 con CSS

Tabla de contenidos

Tema 03.2—

Diseñoweb

Introducción

1. Cajas 5
Display 6
2. Modelo de caja tradicional 7
Contenido flotante 9
Cajas flotantes 14
Posicionamiento absoluto 19
Columnas 26
Aplicación de la vida real 29
3. Modelo de caja flexible 45
Contenedor flexible 46
Elementos flexibles 47
Organizando elementos flexibles 55
Aplicación de la vida real 69

Tema 03.2

Diseño web

Introducción

El diseño web con CSS es una técnica para dar estilo y formato a las páginas web. La ubicación de los bloques en una página web se puede lograr utilizando diferentes técnicas de posicionamiento en CSS, como posición estática, relativa, absoluta y fija.

Un buen diseño web es importante por varias razones:

- Mejora la experiencia del usuario: Un diseño atractivo, intuitivo y fácil de usar mejora la experiencia del usuario en un sitio web y aumenta la probabilidad de que regresen.
- Aumenta la credibilidad: Un sitio web bien diseñado y profesional transmite una imagen positiva y aumenta la confianza y credibilidad en la marca o empresa.
- Mejora la accesibilidad: Un buen diseño web asegura que el contenido sea accesible para todos, incluyendo personas con discapacidades y usuarios en dispositivos móviles.
- Aumenta la conversión: Un diseño atractivo y bien organizado puede aumentar la tasa de conversión y mejorar el rendimiento de un sitio web en términos de ventas o generación de leads.
- Facilita la navegación: Un buen diseño web permite a los usuarios navegar y encontrar información de manera fácil y rápida, lo que mejora su experiencia en el sitio.

En resumen, un buen diseño web es esencial para el éxito en línea y para lograr los objetivos de un sitio web, ya sea para una empresa, una marca o un individuo.

Tema 03.2
Diseño web

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 3.2-1.

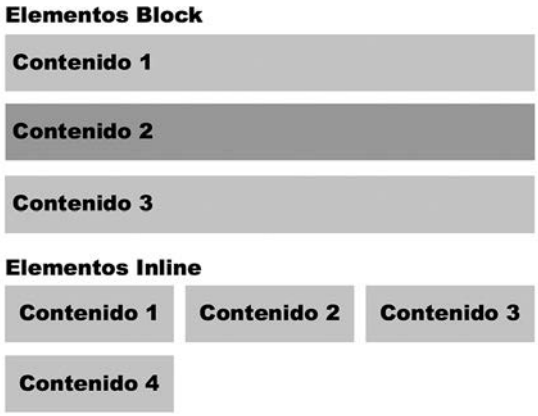


Figura 3.2-1: Elementos *Block e Inline*

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 , se declaran como elementos Block por defecto, y otros como , , o , 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 3.2-1: *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.

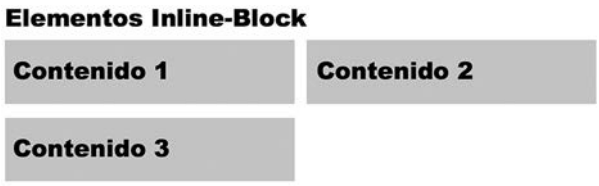


Figura 3.2-2: *Elementos Inline-Block*

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.

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 3.2-2: *Probando la propiedad* `float`

El documento del Listado 3.2-2 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>.

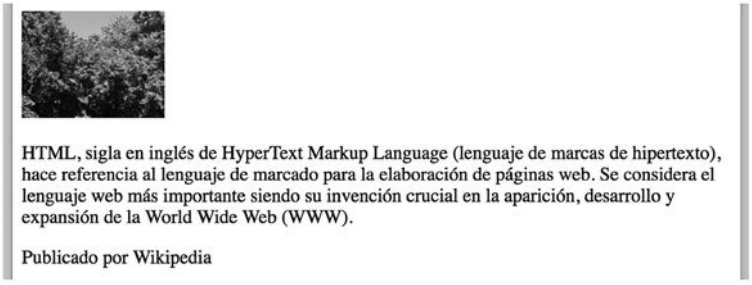


Figura 3.2-3: *Imagen y párrafo con estilos por defecto*

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 3.2-3: *Flotando la imagen hacia la izquierda*

Cuando un elemento flota hacia un lado, los siguientes elementos flotan a su alrededor ocupando el espacio disponible.

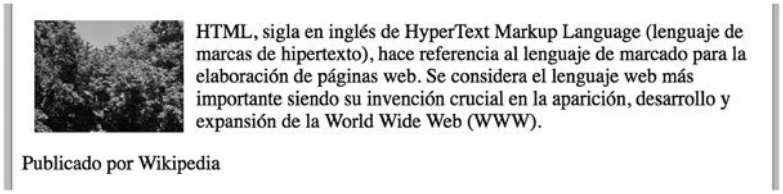


Figura 3.2-4: *Imagen flota hacia la izquierda*

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.

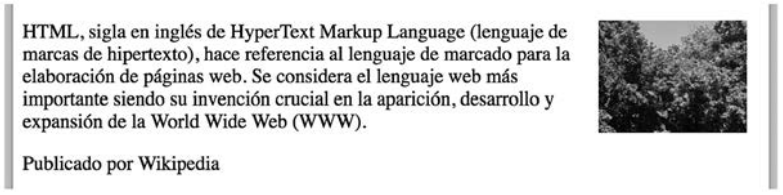


Figura 3.2-5: *Imagen flota hacia la derecha*

Ejercicio 1: cree un nuevo archivo HTML con el documento del Listado 3.2-2 y un archivo CSS con el nombre misestilos.css y la regla del Listado 3.2-3. 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 3.2-4. 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 3.2-5.

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).

Figura 3.2-6: *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 3.2-6 (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 3.2-4: *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.

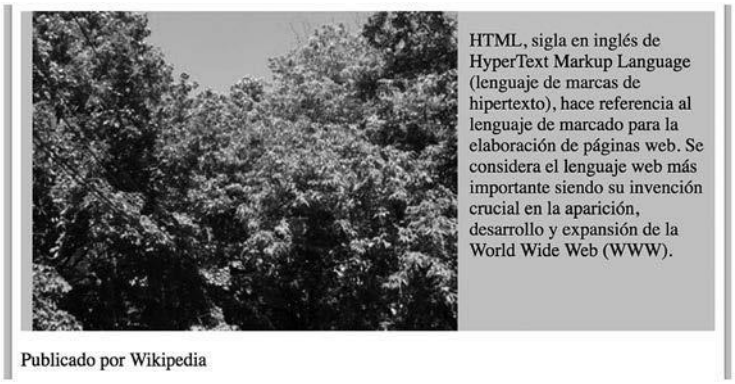


Figura 3.2-7: *La imagen ya no desborda a su contenedor*

Ejercicio 2:elimine el atributo `width` del elemento `<img>` en el documento HTML del Listado 3.2-2. Reemplace las reglas en su archivo CSS porel código del Listado 3.2-4. Abra el documento en su navegador. Debería ver algo parecido a la Figura 3.2-7.

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 `<div>` 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. </p>
     <div class="clearfloat"></div>
   </section>
   <footer>
     <p>Publicado por Wikipedia</p>
   </footer>
 </body>
 </html>
```

Listado 3.2-5: *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 3.2-6: *Restaurando el flujo normal del documento con la propiedad* `clear`

Con las reglas del Listado 3.2-6, 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 3.2-7.

Ejercicio 3:actualice su archivo HTML con el documento del Listado 3.2-5 y reemplace las reglas en su archivo CSS por el código del Listado 3.2-6. Abrael documento en su navegador. Debería ver algo similar a la Figura 3.2-7.

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 3.2-7: *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 3.2-8: *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 perode 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 3.2-8: *Flotando cajas a la izquierda*

En el ejemplo del Listado 3.2-8, 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 3.2-9: *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.

Ejercicio 4:cree un nuevo archivo HTML con el código del Listado 3.2- 7 y un nuevo archivo CSS llamado misestilos.css con las reglas del Listado 3.2-8.Abra el documento en su navegador. Debería ver algo similar a la Figura 3.2-9.

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 3.2-9: *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.

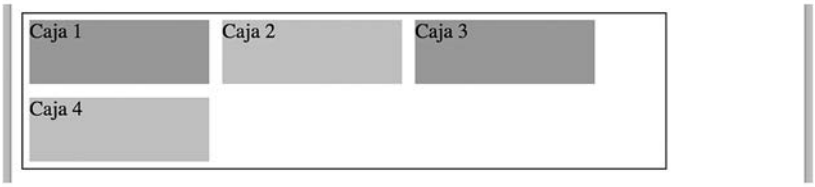


Figura 3.2-10: *Las cajas llenan el espacio disponible*

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 3.2-10: *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 3.2-11: *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 3.2-7 y mover el elemento **`<div>`** identificado con el nombre **`caja-4`** sobre el elemento **`<div>`** 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 3.2-11: *Especificando la posición relativa de un elemento*

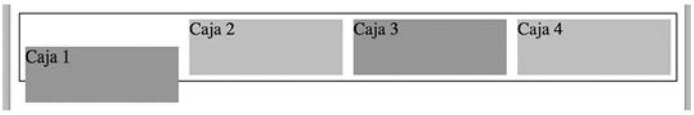


Figura 3.2-12: *Posición relativa*

Ejercicio 5:reemplace la regla **`caja-1`** en su hoja de estilo CSS por laregla del Listado 3.2-11. 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 3.2-12: *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 3.2-13: *Posición absoluta*

Ejercicio 6:reemplace las reglas **`#cajapadre`** y **`#caja-1`** en su hoja de estilo CSS con las reglas del Listado 3.2-12 y abra el documento en su navegador. Debería ver algo parecido a la Figura 3.2-13.

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 3.2-13: *Especificando el índice z*

Los índices negativos se consideran más bajos que los índices asignados por defecto. En el Listado 3.2-13, 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 3.2-14: 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 3.2-14: *Mostrando una etiqueta desplegable con información adicional*

El documento del Listado 3.2-14 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 3.2-15 ilustra la estructura generada por estos elementos.

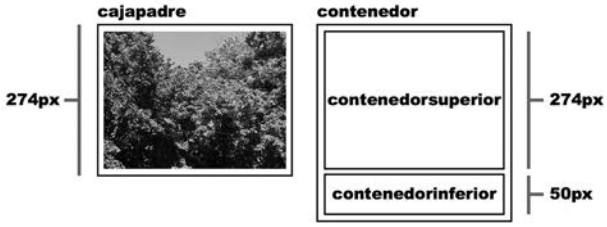


Figura 3.2-15: *Estructura para presentar una etiqueta desplegable con información adicional*

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 3.2-15: *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 3.2-15 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 3.2-16: *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 3.2-16: *Animando la etiqueta*

Ejercicio 7:cree un nuevo archivo HTML con el código del Listado 3.2-14 y un archivo CSS llamado misestilos.css con el código del Listado 3.2-15. 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 3.2-16). Reemplace la regla **`#contenedor`** de su hoja de estilo con la regla del Listado 3.2-16. 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. </span>
    </article>
  </section>
</body>
</html>
```

Listado 3.2-17: *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 3.2-18: *Definiendo las columnas*

La regla del Listado 3.2-18 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 3.2-17.

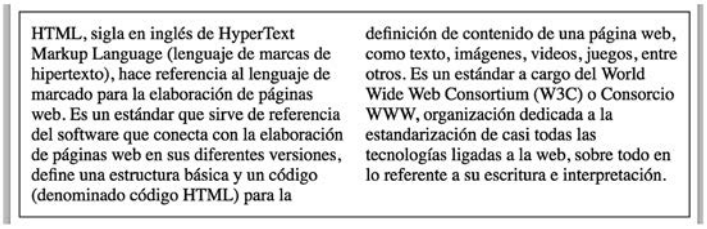

HTML, sigla en inglés de HyperText Markup Language (lenguaje de marcas de hipertexto), hace referencia al lenguaje de marcado para la elaboración de páginas web. Es un estándar que sirve de referencia del software que conecta con la elaboración de páginas web en sus diferentes versiones, define una estructura básica y un código (denominado código HTML) para la definición de contenido de una página web, como texto, imágenes, videos, 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.

Figura 3.2-17: *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.

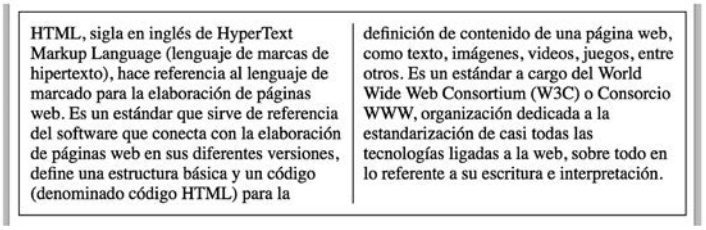
column-rule—Esta propiedad nos permite definir todos los valores de la línea al mismo tiempo.

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 3.2-19: *Agregando una línea entre las columnas*



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, videos, 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.

Figura 3.2-18: *Columnas separadas por una línea*

Ejercicio 8:cree un nuevo archivo HTML con el código del Listado 3.2-17 y un archivo CSS llamado misestilos.css con el código del Listado 3.2-18. Abra el documento en su navegador. Debería ver el texto dividido en dos columnas. Reemplace la regla **`#articulosnoticias`** por la regla del Listado 3.2-19 y actualice la página en su navegador. Debería ver una línea dividiendo las columnas, tal como muestra la Figura 3.2-18.

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 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 3.2-20: *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 adicional. Ahora podemos asignar tamaños y alineamientos independientes para cada sección del documento.

Ejercicio 9:cree un nuevo archivo HTML con el documento del Listado 3.2-20 y un archivo CSS llamado misestilos.css para incluir todos los códigos CSS que vamos a presentar a continuación. Abra el documento en sunavegador 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 3.2-21: *Implementando una regla de reseteado*

La primera regla de nuestro archivo CSS introducida en el Listado 3.2-21 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.

Ejercicio 10:para crear la hoja de estilo para el documento del Listado 3.2-20, tiene que incluir todas las reglas presentadas desde el Listado 3.2-21, una sobre otra en el mismo archivo (misestilos.css).

Lo básico: lo que hemos hecho con la regla del Listado 3.2-21 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 3.2-19: *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 3.2-22: *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 que son descendientes directos del elemento <header>. Como solo tenemos un único elemento 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 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 3.2-20.

Figura 3.2-20: *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 3.2-23: *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 3.2-24: *Asignando estilos a las opciones del menú*

La primera regla del Listado 3.2-24 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 3.2-21.

Figura 3.2-21: *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 3.2-25: *Asignando estilos al contenido principal*

Las reglas del Listado 3.2-25 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 3.2-25.

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 3.2-26: *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.

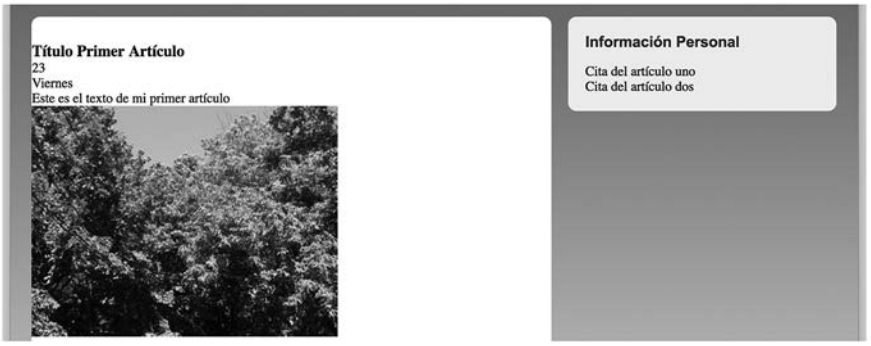


Figura 3.2-22: *Contenido principal*

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 3.2-26 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 3.2-27: *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 3.2-27. 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 3.2-23: *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 3.2-28: *Asignando estilos a los artículos*

Figura 3.2-24: *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 3.2-29: *Asignando estilos al pie de página*

Las reglas del Listado 3.2-29 definen tres secciones de 276 píxeles cada una, todas flotando hacia la izquierda para formar las tres columnas necesarias para nuestro diseño.

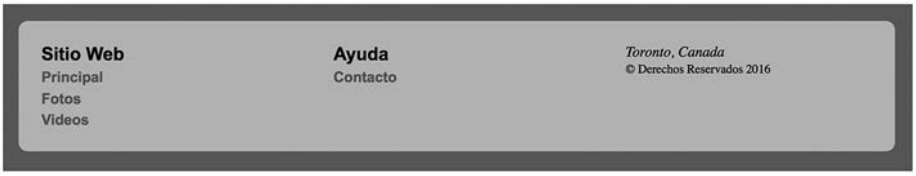


Figura 3.2-25: *Pie de página*

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 3.2-30: *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 3.2-30 a nuestro documento, el pie de página se mostrará parecido al de la Figura 3.2-25, 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.

Ejercicio 11:si aún no lo ha hecho, asigne el nombre index.html al documento del Listado 3.2-20. 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 3.2-31: *Agregando submenús*

Los estilos para un menú que incluye submenús 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 3.2-31, 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 3.2-32: *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 3.2-26: *Submenús*

Ejercicio 12:actualice su documento con el código del Listado 3.2-31 y las reglas de su hoja de estilo con el código del Listado 3.2-32. Abra el archivo index.html en su navegador y mueva el ratón sobre la opción **Fotos**. Debería ver algo similar a la Figura 3.2-26.

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.

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 3.2-33: *Organizando cajas con un contenedor flexible*

El documento del Listado 3.2-33, 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 3.2-34: *Convirtiendo el elemento* `cajapadre` *en un contenedor flexible*

Ejercicio 12:cree un nuevo archivo HTML con el código del Listado 3.2-33 y un archivo CSS llamado misestilos.css con la regla del Listado 3.2-34. 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 3.2-35: *Convirtiendo las cajas en cajas flexibles con la propiedad* `flex`

En el ejemplo del Listado 3.2-35, 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).

Figura 3.2-27: *Los mismos valores para la propiedad* `flex` *distribuyen el espacio equitativamente*

Ejercicio 13:reemplace las reglas en su archivo CSS por el código del Listado 3.2-35 y abra el documento del Listado 3.2-33 en su navegador. Debería ver algo parecido a la Figura 3.2-27.

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 3.2-36: *Creando una distribución desigual*

En el Listado 3.2-36, 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.

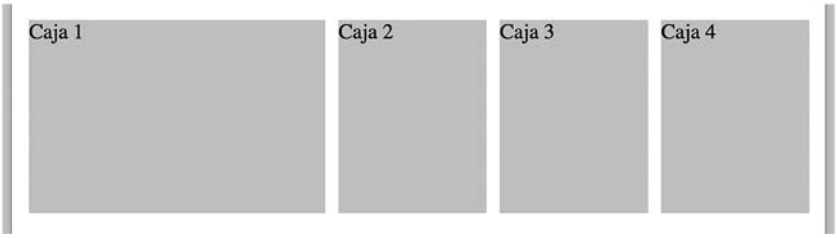


Figura 3.2-28: *Distribución desigual con* `flex`

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 3.2-37: *Combinando cajas flexibles con cajas inflexibles*

La primera caja del ejemplo del Listado 3.2-37 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.

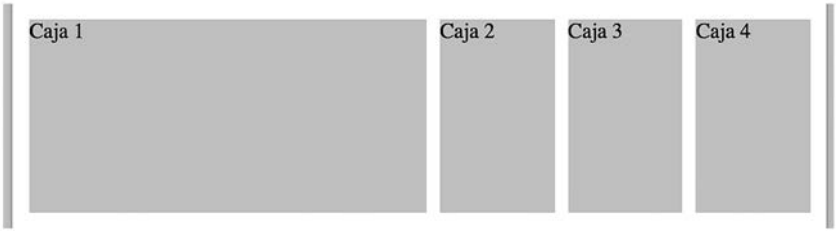


Figura 3.2-29: *Solo se distribuye el espacio libre*

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 3.2-38: *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.

Ejercicio 14 :reemplace las reglas de su archivo CSS por el código delListado 3.2-38. 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 3.2-39: *Definiendo cajas flexibles con un tamaño preferido*

En el Listado 3.2-39, 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 3.2-30: *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 3.2-40: *Declarando un tamaño máximo*

Figura 3.2-31: *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.

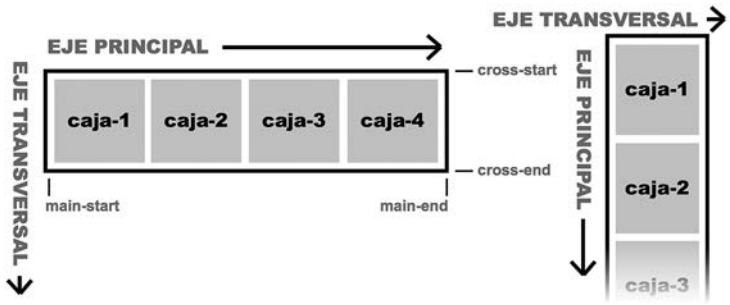


Figura 3.2-32: *Ejes de contenedores flexibles*

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 3.2-32, 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 3.2-41: *Invirtiendo la orientación de las cajas*

Figura 3.2-33: *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 3.2-42: *Definiendo la posición de cada caja*

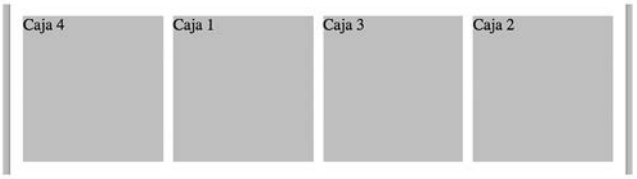


Figura 3.2-34: *Nuevas posiciones para cada caja definidas por la propiedad* `order`

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 3.2-43: *Distribuyendo el espacio libre en un contenedor flexible*

El ejemplo del Listado 3.2-43 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.

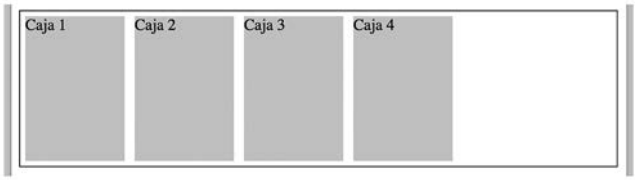


Figura 3.2-35: *Cajas y espacio libre dentro de un contendor flexible*

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 3.2-35, 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 3.2-44: *Distribuyendo el espacio vacío con la propiedad* `justify-content`

Figura 3.2-36: *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 3.2-37: *Espacio vacío distribuido con* `justify-content: center`

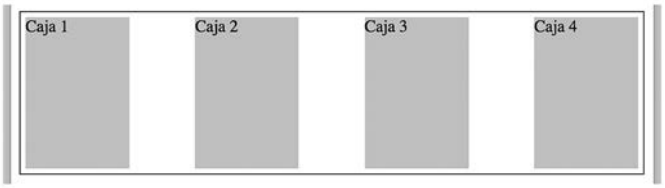


Figura 3.2-38: *Espacio vacío distribuido con* `justify-content: space-between`

Figura 3.2-39: *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 apropiada 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 3.2-45: *Distribuyendo el espacio vertical*

En el Listado 3.2-45 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 3.2-40.

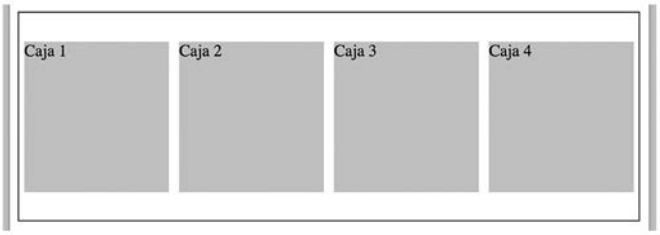


Figura 3.2-40: *Alineación vertical con* `align-items: center`

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 3.2-46: *Estirando las cajas para ocupar el espacio vertical disponible*

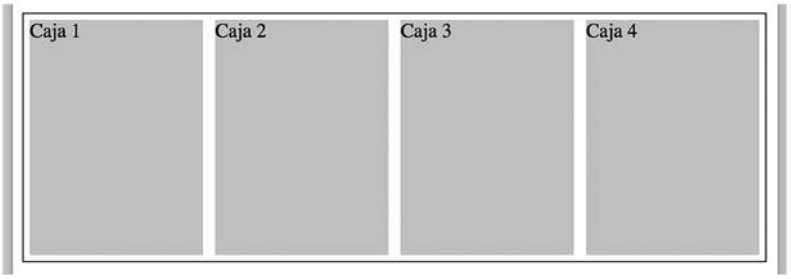


Figura 3.2-41: *Estirando las cajas con* `align-items: stretch`

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 3.2-47: *Alineando las cajas hacia el extremo superior*

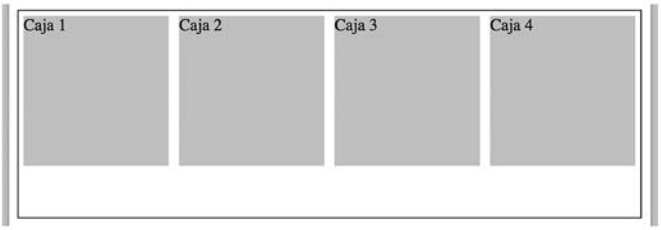


Figura 3.2-42: *Cajas alineadas con* `align-items: flex-start`

El valor flex-end alinea las cajas hacia el final del contenedor (normalmente el extremo derecho o inferior).

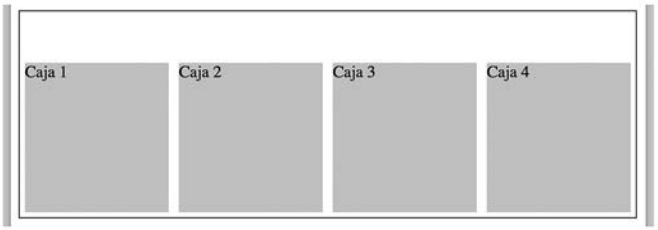


Figura 3.2-43: *Cajas alineadas con* `align-items: flex-end`

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 3.2-48: *Alineando las cajas por la línea base*

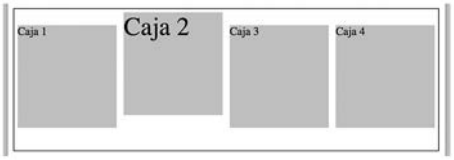


Figura 3.2-44: *Cajas alineadas con* `align-items: baseline`

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 3.2-49: *Cambiando la alineación del elemento* `caja-2`

Las reglas del Listado 3.2-49 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.

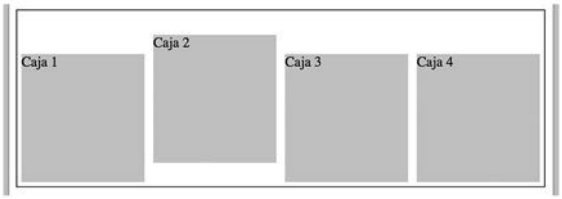


Figura 3.2-45: *Caja alineada con* `align-self`

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 3.2-50: *Creando dos líneas de cajas con la propiedad* `flex-wrap`

En el Listado 3.2-50, 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 3.2-46.

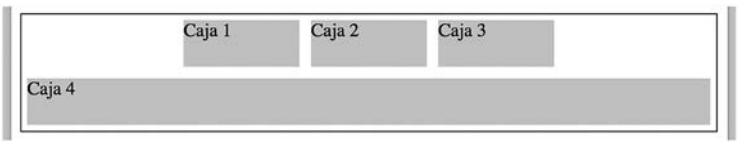


Figura 3.2-46: *Múltiples líneas en un contenedor flexible*

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.

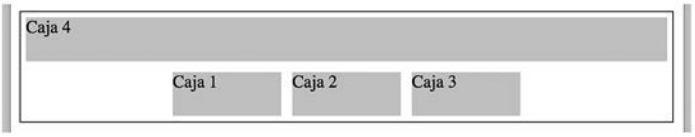


Figura 3.2-47: *Nuevo ordenamiento de líneas usando* `flex-wrap: wrap-reverse`

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 3.2-51: *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 3.2-51, 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.

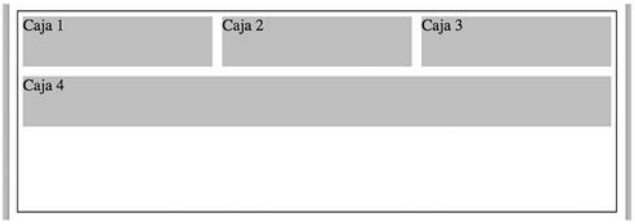


Figura 3.2-48: *Líneas alineadas con* `align-content: flex-start`

Las siguientes figuras muestran el efecto producido por el resto de los valores disponibles para la propiedad align-content.

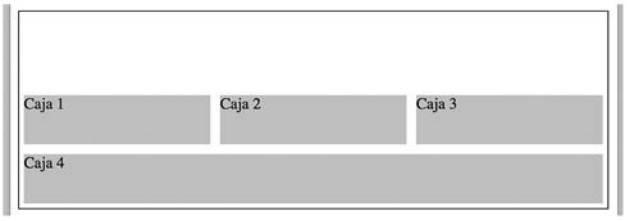


Figura 3.2-49: *Líneas alineadas con* `align-content: flex-end`

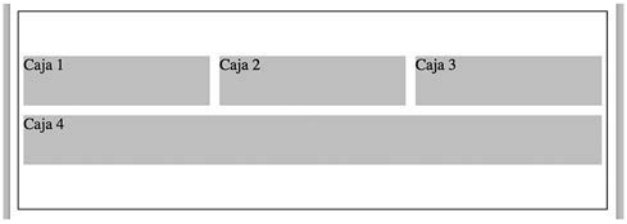


Figura 3.2-50: *Líneas alineadas con* `align-content: center`

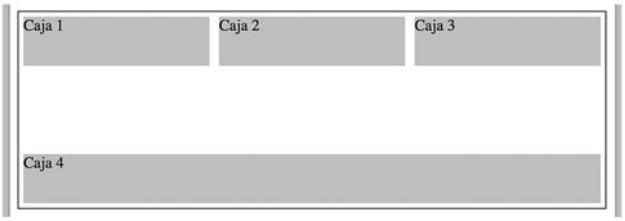


Figura 3.2-51: *Líneas alineadas con* `align-content: space-between`

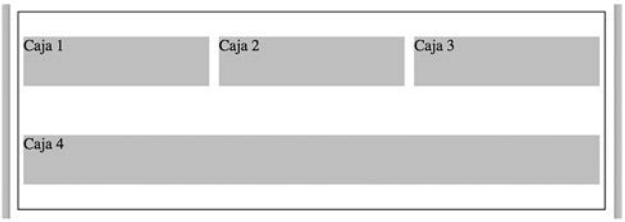


Figura 3.2-52: *Líneas alineadas con* `align-content: space-around`

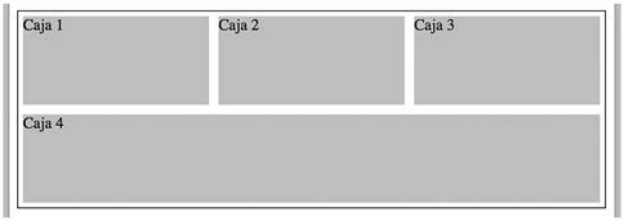


Figura 3.2-53: *Líneas alineadas con* `align-content: stretch`

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 3.2-20, 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 3.2-52: *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 3.2-53: *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 3.2-54: *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 3.2-55: *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 3.2-56: *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 3.2-57: *Convirtiendo al pie de página en un contenedor flexible*

Ejercicio 15 :cree un nuevo archivo HTML con el código del Listado 3.2-52 y un archivo CSS llamado misestilos.css con las reglas introducidas desde el Listado 3.2-53. 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.