CHAPTER 7

Page Layouts

When you complete this chapter, you will be able to:

- Understand the normal flow of elements
- Use the division element to create content containers
- Create floating layouts
- Build a flexible page layout
- Build a fixed page layout

In a standard HTML document, the default positioning of elements is generally top to bottom and left to right. In the past, Web designers used tables to create multiple column layouts and gain more control of their page designs. HTML tables are not intended for page layout and are no longer in favor, although they still exist on many Web pages.

Modern Web designs are built using the CSS layout capabilities. As you saw in Chapter 6, you can use floats to position content elements on a Web page and move them out of the normal flow of elements. In this chapter, you will learn how to expand on this concept by using floats to create multicolumn Web pages that can either be flexible based on the browser size and screen resolution, or fixed to a definite width. You will see how to resolve common float problems, and you will get a chance to build both a flexible and a fixed page layout.

Understanding the Normal Flow of Elements

By default, the browser normally displays elements on the page one after the other, depending on whether the elements are block-level or inline elements. Some elements float to the left or right of other elements on the page, as you saw in Chapter 6. Element position can be affected by margin or padding properties, but generally the browser lays out element boxes from top to bottom and left to right until all elements that make up the Web page have been displayed.

In the normal flow for block-level elements, boxes are laid out vertically one after the other, beginning at the top of the containing block. Each box horizontally fills the browser window. The space between boxes is determined by the margin settings. The **normal flow** determines the sequence of element display with which you are familiar in standard HTML. For an example of normal flow, examine the following HTML code and the resulting normal flow diagram in Figure 7-1:

```
<body>
   <h1>The Document Title</h1>
   <p>Lorem ipsum...</p>
   <p>Duis autem...</p>
   <p>Ut wisi enim...</p>
</body>
```

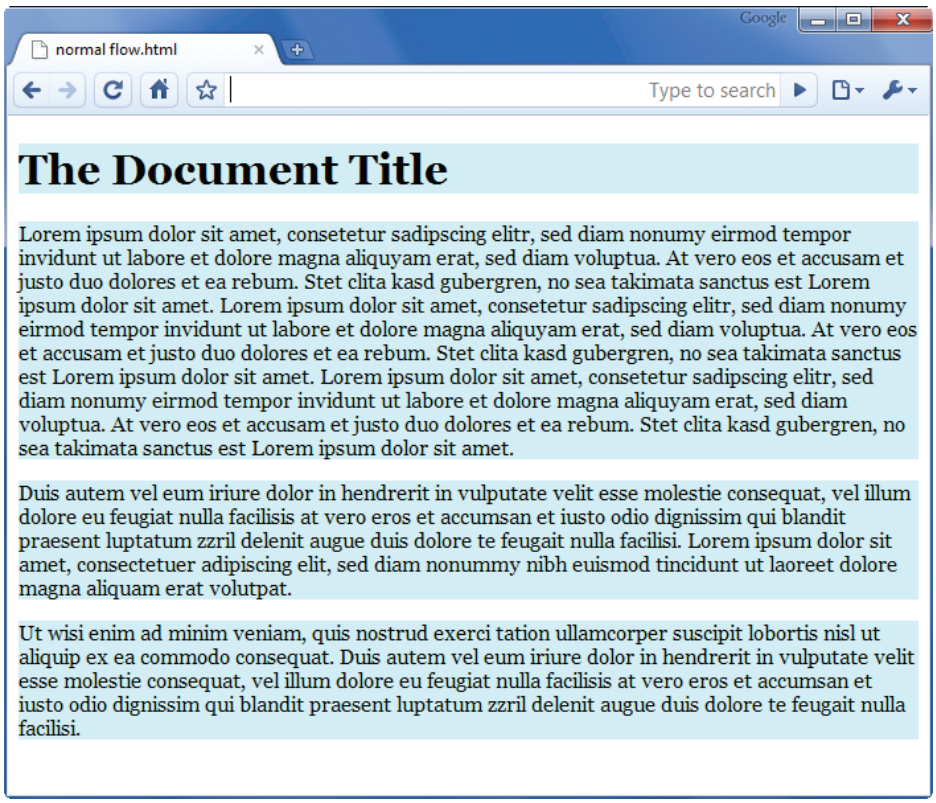


Figure 7-1 Block-level element normal flow

The <body> element is the containing element for the content section of the Web page. The elements within the <body> element are displayed exactly in the order they appear in the code from top to bottom, which in this example is an <h1> followed by three <p> elements. Elements do not appear next to each other unless they are floated (see Chapter 6) or have a display type of *inline*.

In the normal flow for inline elements, boxes are laid out horizontally, beginning at the top left of the containing block. The inline boxes comprise the lines of text within, for example, a <p> element. The browser flows the text into the inline boxes, wrapping the text to the next line box as determined by the constraints of the containing box or the browser window size.

When you **float**, or position an element, you take it out of the normal flow. Other elements that are not floated or positioned will still follow the normal flow, so you should check the results frequently as you are designing your layout using floats. Figure 7-2 shows a floated element on the left side of the page. Notice that

the two other nonfloating elements remain in the normal flow and still span the width of the browser window, even extending behind the floating elements. As you can see in the example, the text in the normal flow boxes appears in the correct position to the right of the floated element. This behavior allows text to wrap around floated elements such as images, which is a basic advantage of floats. However, when you start to use floats to build page layouts, the behavior of floats can cause problems. As you gain more experience working with floating layouts, you will be able to anticipate and correct problems with floats as you code your page designs.

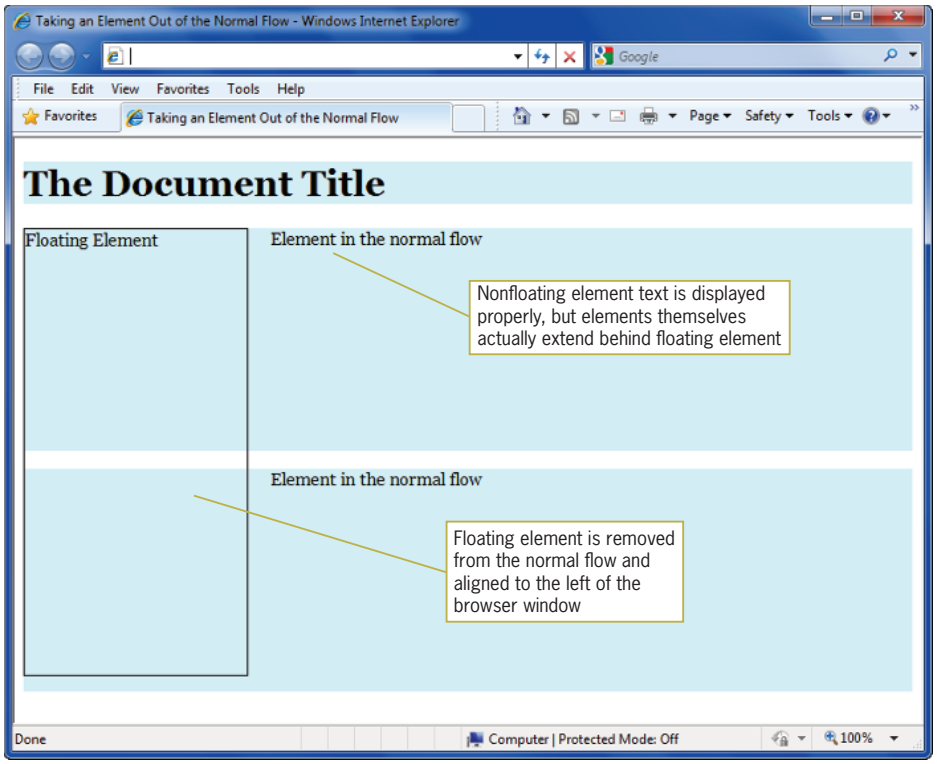


Figure 7-2 Floating element is removed from the normal flow

Using the Division Element to Create Content Containers

The division element is your primary tool for creating sections of content in your Web page designs. Using the box properties you learned about in Chapter 6, you can create divisions that are any

shape you need to contain and segregate sections of content. You can create vertical columns containing content and control the white space between and within the columns. You can nest divisions within divisions and create interesting content presentations. Finally, you can create a division element to contain an entire Web page, often called a **wrapper**, to center a Web page within the browser window, regardless of screen resolution.

Figure 7-3 shows a Web page created with multiple divisions. The wrapper division, outlined in red, contains all of the content of the Web page. The wrapper has three child division elements; header, navigation, and article. Each of these divisions will contain page content.

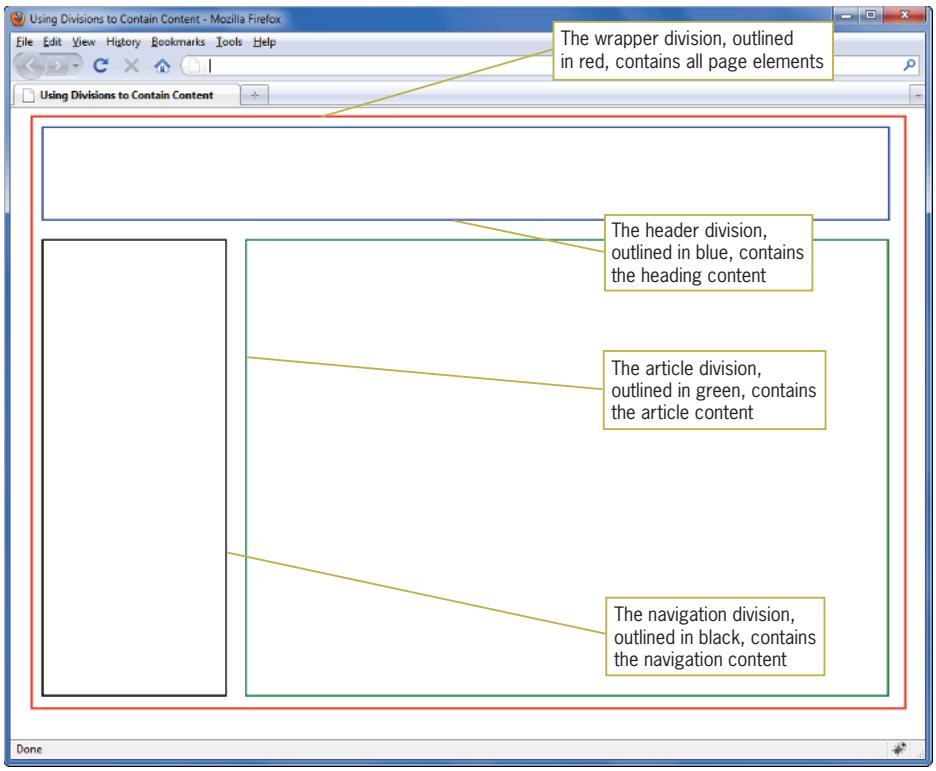


Figure 7-3 Web page with multiple content divisions

In Figure 7-4, you can see these same divisions with sample content. The banner division contains an <h1> element. The navigation division contains an <h2> heading and links. The main division contains an <h2> heading, an image and paragraph elements. Various margin and padding settings offset the content

from the sides of the container elements. The wrapper element holds all the pieces together, and lets the page be centered in the browser window.

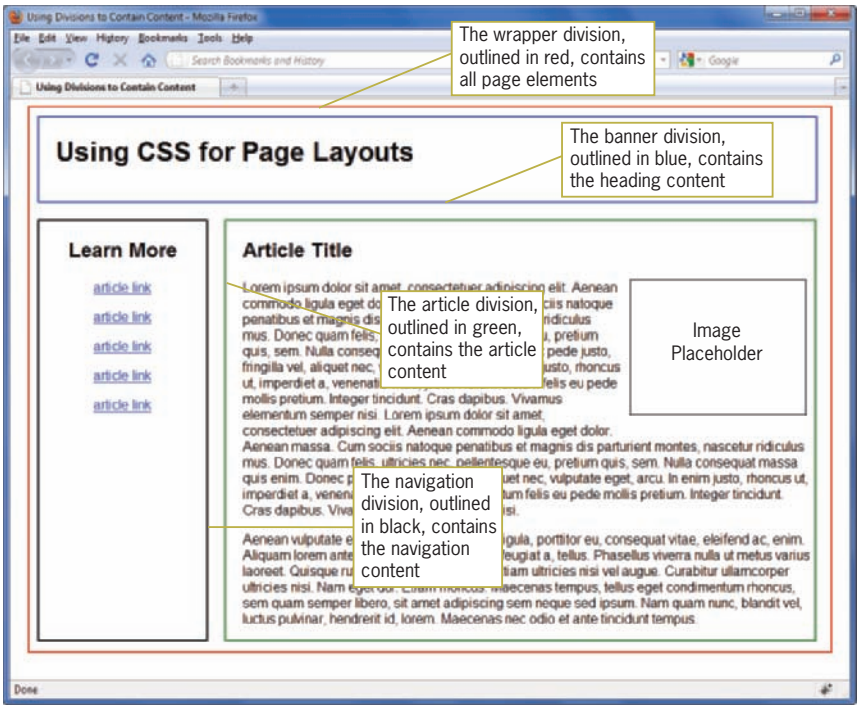


Figure 7-4 Web page with multiple content divisions, filled with content

Creating Floating Layouts

The float property lets you build columnar layouts by floating content elements to either the right or left side of the browser window. A typical Web page design can contain both floating and nonfloating elements. For example, Figure 7-5 shows a Web page layout with a header, three columns of content, and a footer. The three columns are floating divisions, while the header and footer are nonfloating. The nav and article columns are floating to the left, while the sidebar is floating to the right. All of the page elements are separated from each other with margin settings to provide gutters between the columns.

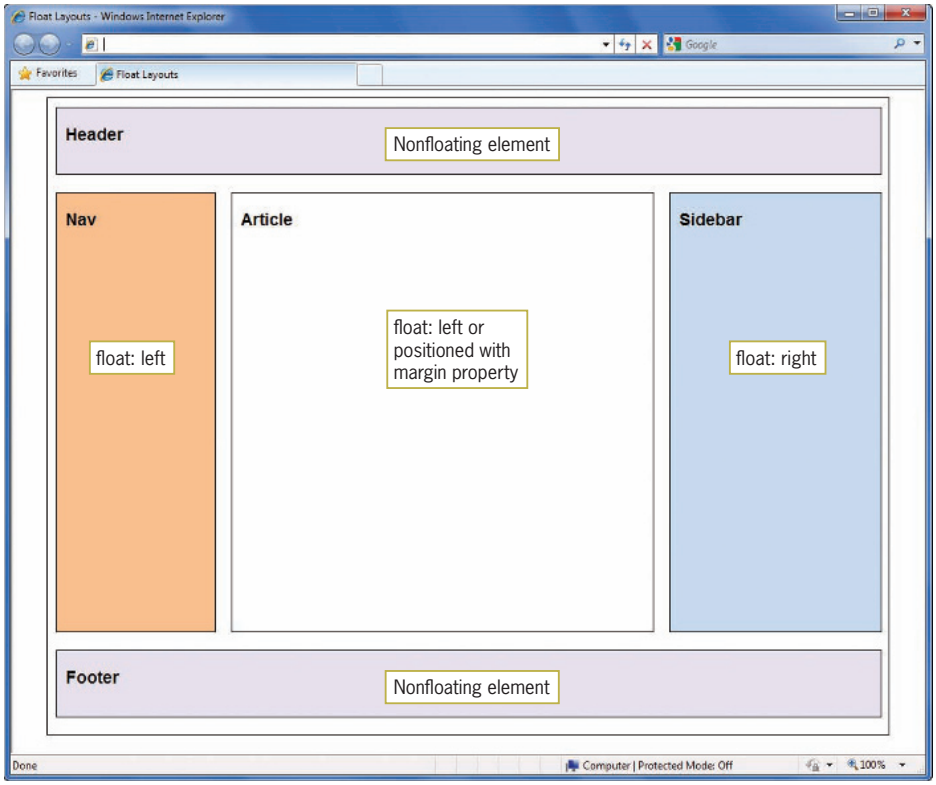


Figure 7-5 Floating and nonfloating elements make up a Web page layout

Building floating layouts requires that you choose a method for containing the floating elements. Floats are designed to extend outside of their containing element. This is because the original concept of floating was to allow text to wrap around images, as you saw earlier in this chapter. When you start to build floating layouts, you will often see that the floating elements extend beyond their containing elements, which will result in a "broken" layout as illustrated in Figure 7-6.

Floating elements must always have a specified width or they will expand to the size of the browser window.

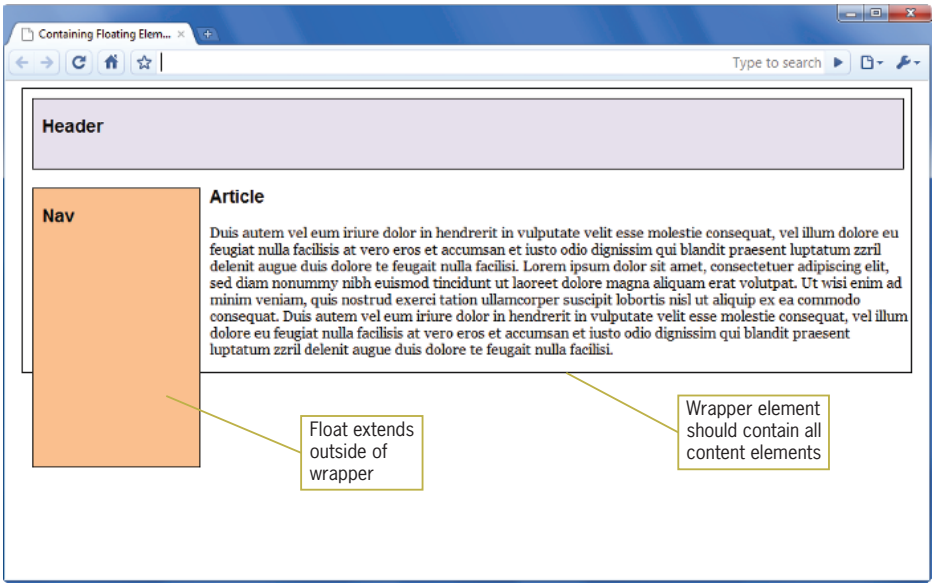


Figure 7-6 Floating element extends outside of containing element

There are two methods you can use to fix this problem.

Solution 1: Using a Normal Flow Element

If you have multiple columns, at least one needs to be nonfloating (in the normal flow), and positioned with margin properties, as shown in Figure 7-7.

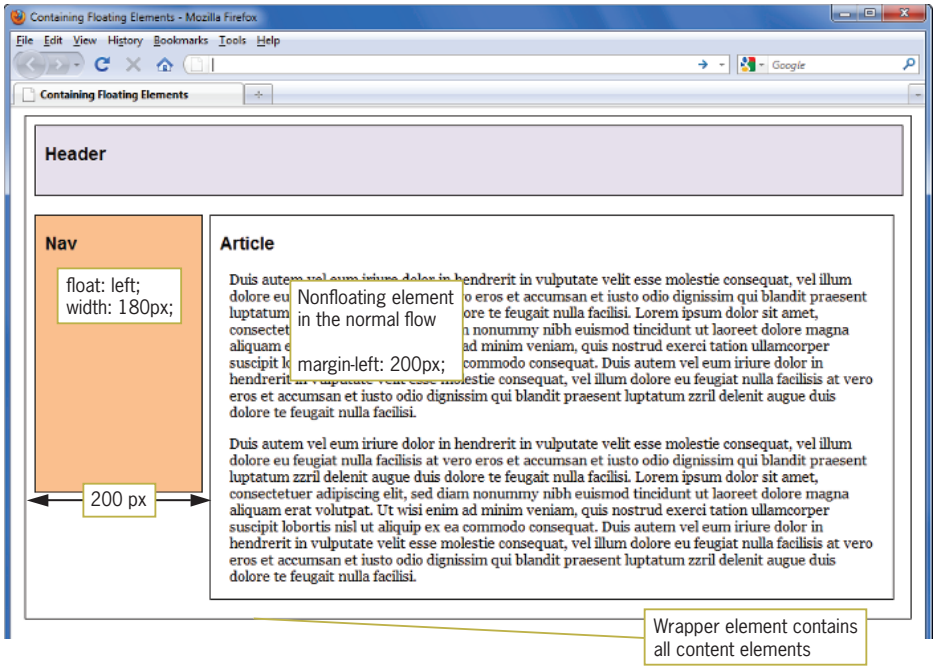


Figure 7-7 Using a normal flow element to contain floats

In this example, the *article* element is nonfloating and has a 200-pixel left margin to position the element to the right of the floating *nav* element. The style rule for both the *nav* and the *article* elements looks like this:

```
#nav {
   width: 180px;
   height: 300px;
   float: left;
   border: solid thin black;
   margin-top: 20px;
   margin-left: 10px;
   margin-right: 10px;
   background-color: #fabf8f;
}

#article {
   width: 740px;
   margin-left: 200px;
   border: solid thin black;
   background-color: #fff;
   margin-top: 20px;
   margin-bottom: 20px;
}
```

Solution 2: Using the Clear Property

If you use a nonfloating footer element (in the normal flow), with the clear property set to *both*, the containing wrapper will extend to contain all elements, as shown in Figure 7-8.

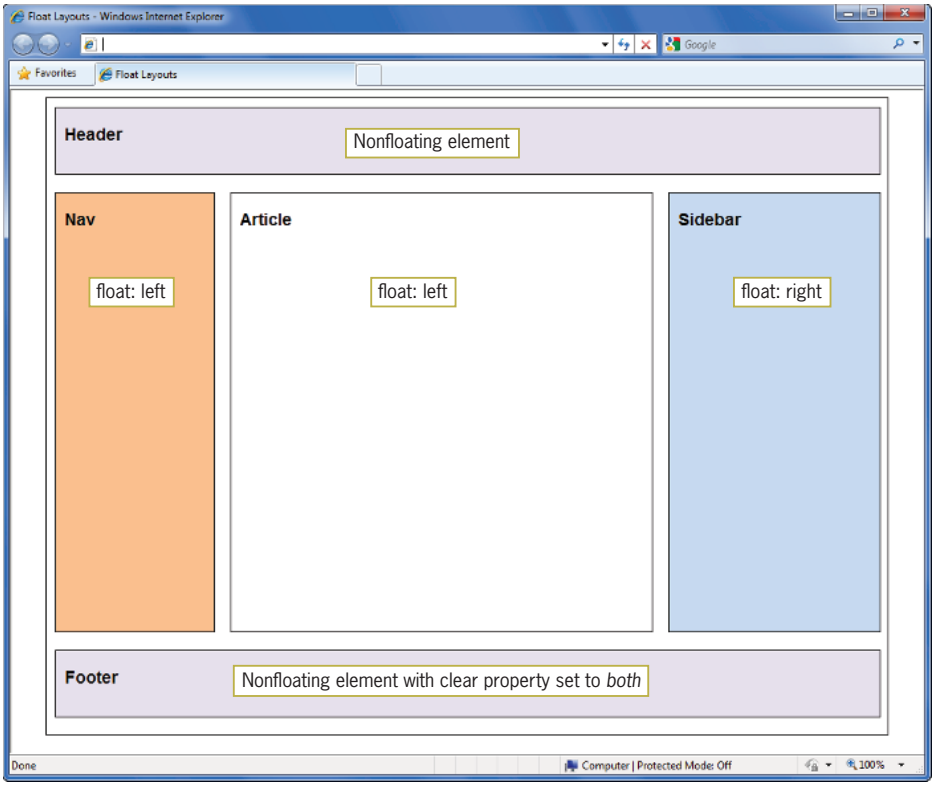


Figure 7-8 Using the clear property to contain floats

In this example, the footer element is nonfloating and has the clear property set to *both*. This forces the wrapper to extend beyond the footer property and contain all of the content elements on the page. The style rule for the footer elements looks like this:

```
#footer {
   width: 940px;
   height: 75px;
   margin-left: 10px;
   clear: both;
   border: solid thin black;
   background-color: #e5dfec;
}
```

Because some type of footer is a consistent design feature in most Web sites, this second solution works very well. Your footer can be as simple as a horizontal rule <hr> element or a graphic contained with a footer division.

Floating Elements Within Floats

Using floating elements gives you a wide variety of options for building interesting page layouts. For example, you can float elements within floating elements, as shown in Figure 7-9.

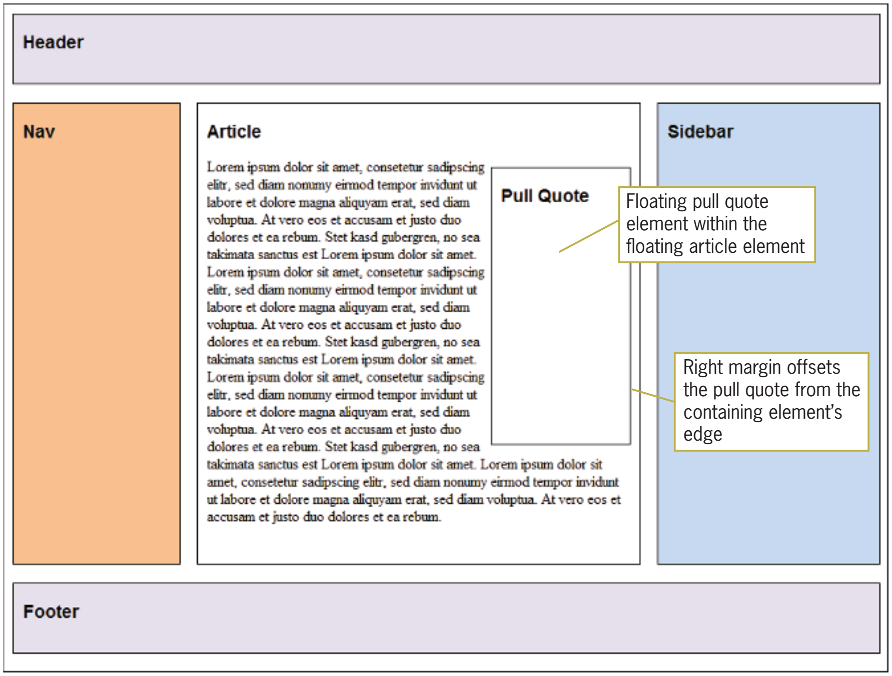


Figure 7-9 Floating element within a float

In this example, a floating element within the article element contains a pull quote, a quotation or excerpt from the main article that would be used to draw the reader's attention to the article. This element floats right within the article element, and is offset from the border of the article element with a right margin. An image element can also be floated using this same method.

When you are floating an element within another element, the order of the elements is important. In the example in Figure 7-10, the floating pull quote element follows the heading element and precedes the paragraph content. If this order is not followed, the

pull quote would appear at the bottom of the article rather than in its correct position. Figure 7-10 shows the order of the elements within the article element.

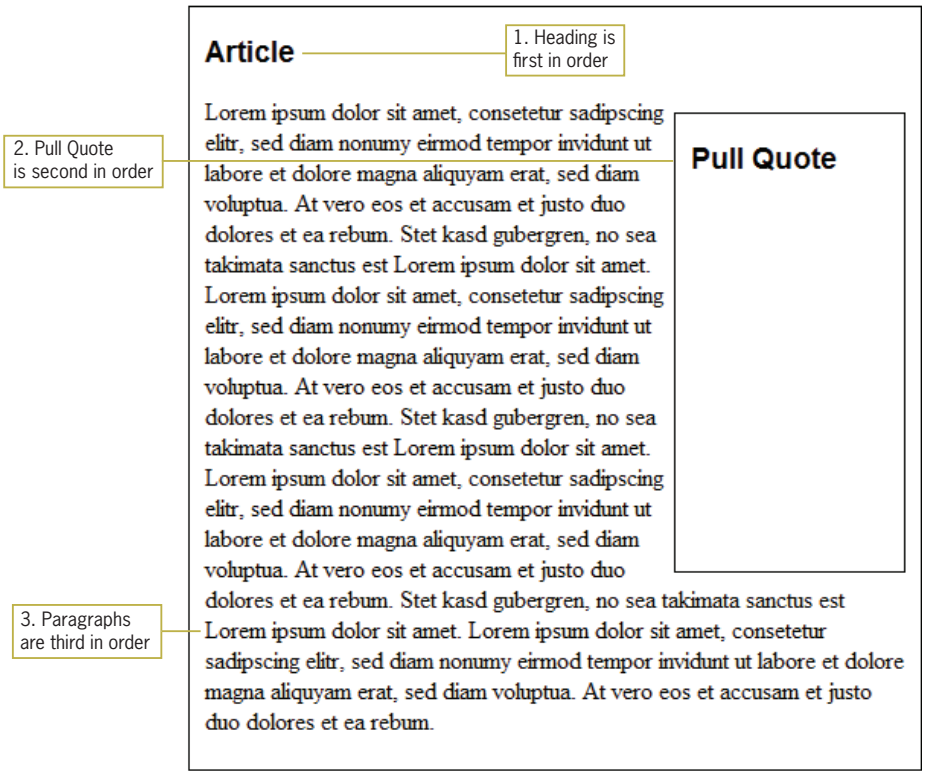


Figure 7-10 Float order is important

In contrast, Figure 7-11 shows what happens to the layout when the correct float order is not maintained. In this example, the order of the pull quote element and the paragraphs has been switched, with unintended results. The pull quote is pushed out of its container element because the paragraph elements take up all of the room in the article element.

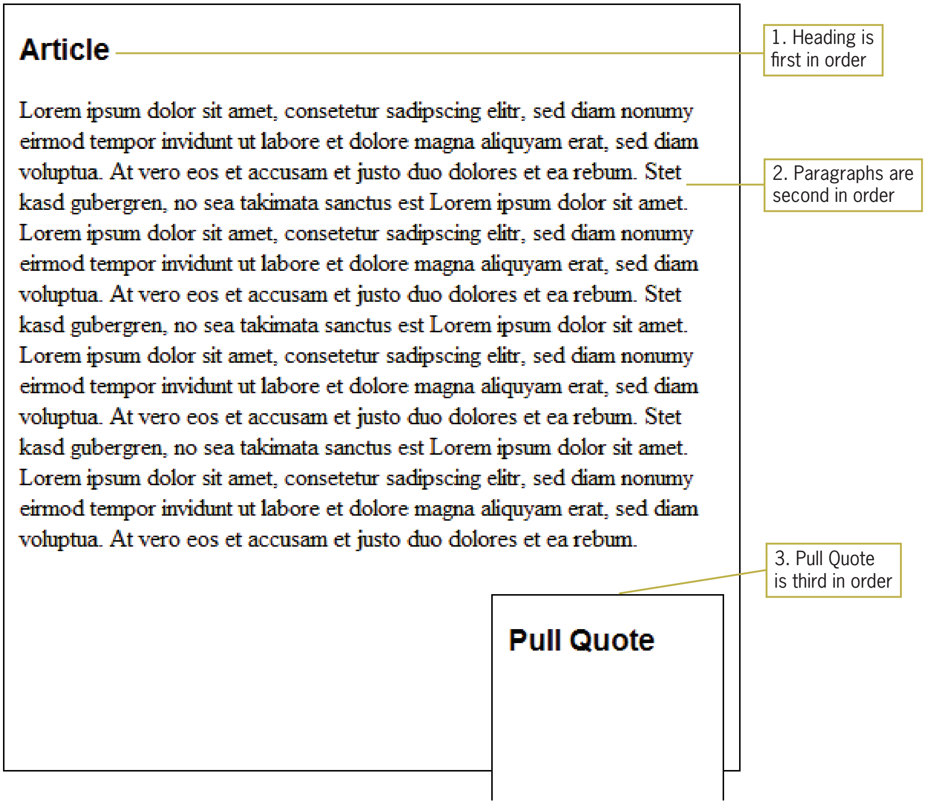


Figure 7-11 Incorrect float order causes layout problems

Fixing Column Drops

Column drops occur when the total width of the columnar elements in a page layout exceeds the width of their containing element. The width of a box element includes the total of its width value plus any left or right padding, border, and margins. Figure 7-12 shows an example of column drop caused by the total width of the contained columns being greater than the width of the containing wrapper element.

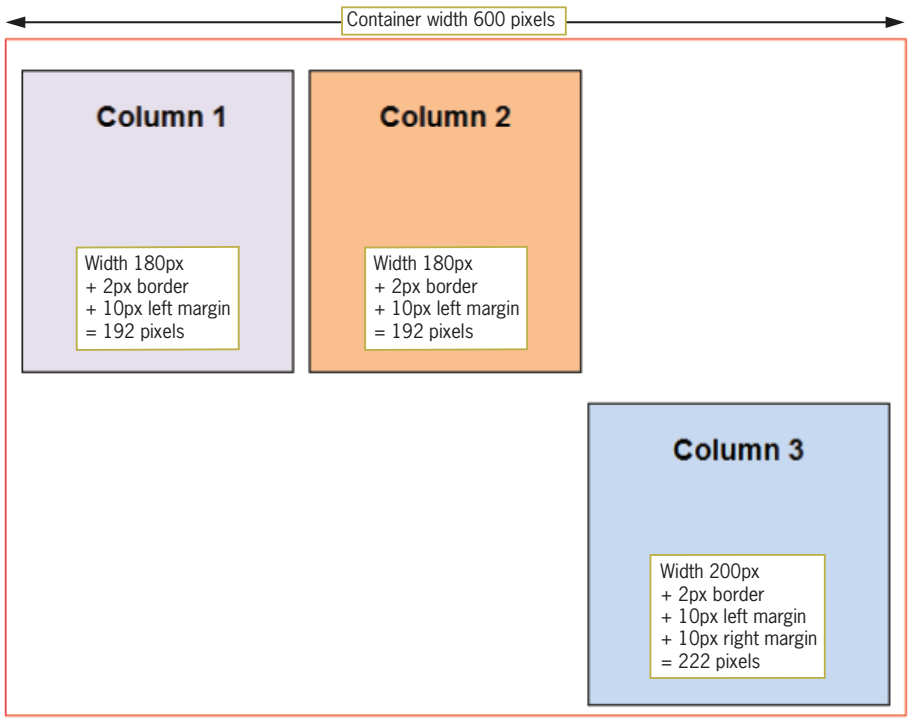
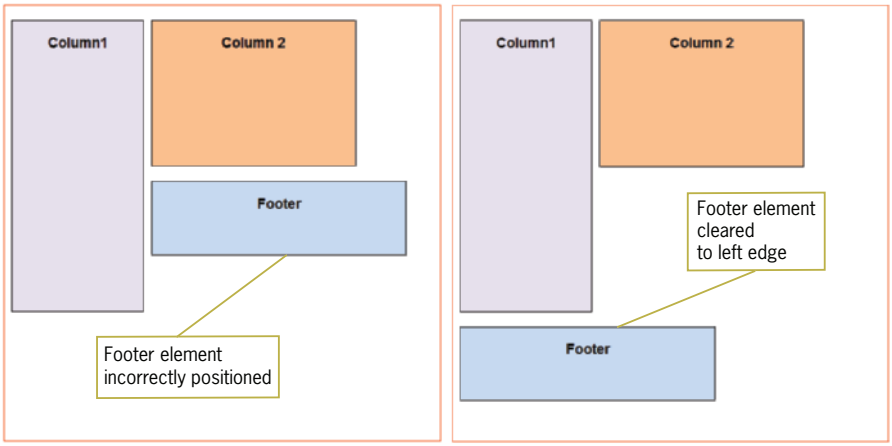


Figure 7-12 Column drop caused by combined element width being greater than container width

If you total the width of the three elements (192 + 192 + 222 = 606), the sum is greater than 600 pixels, the width of the containing element, forcing Column 3 to drop below the other columns because the layout does not provide enough horizontal width. To solve this problem, the width of Column 3 can be reduced to 180 pixels, reducing the overall width of the columns to 586 pixels.

Clearing Problem Floats

When you are designing float-based layouts, floats occasionally do not appear exactly where you want them to appear. The clear property can help you solve this problem. For example, in Figure 7-13a, the footer element floats left and should appear below Column 1, but instead it floats in order after Column 2. To move it down the page to its correct position, add the clear property set to a value of left to move the footer down the page and align it with the next clear left edge of the browser window, as shown in Figure 7-13b.



a. b.

Figure 7-13 Problem floats fixed with the clear property

Building a Flexible Page Layout

Flexible layouts, sometimes known as liquid layouts, adapt to the size of the browser window. Flexible layouts shift as the user resizes the window, wrapping text or adding white space as necessary. In Figure 7-14, the flexible layout contains three elements.

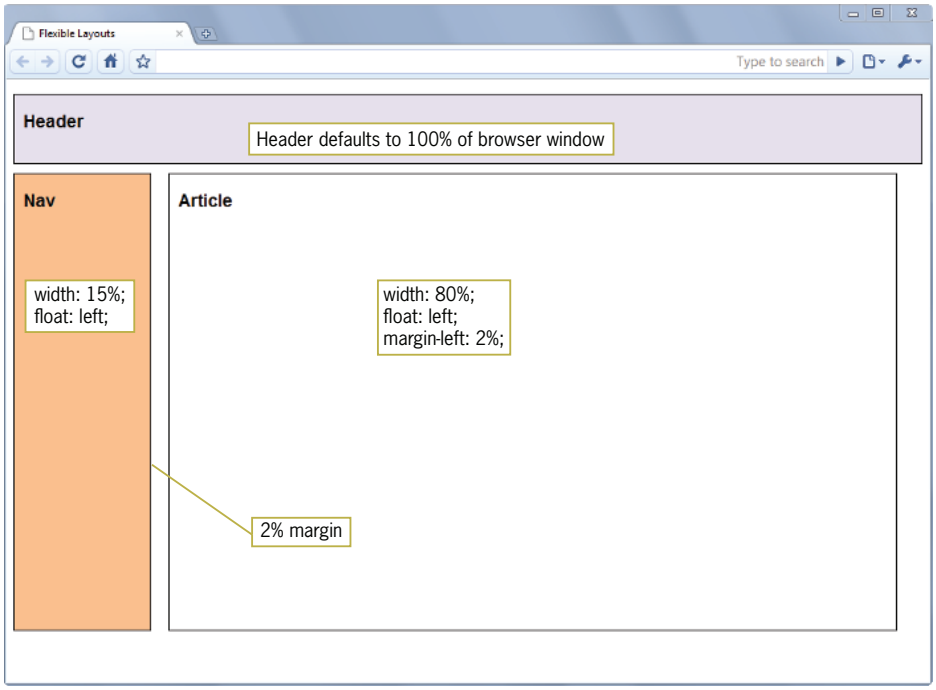


Figure 7-14 Flexible layout

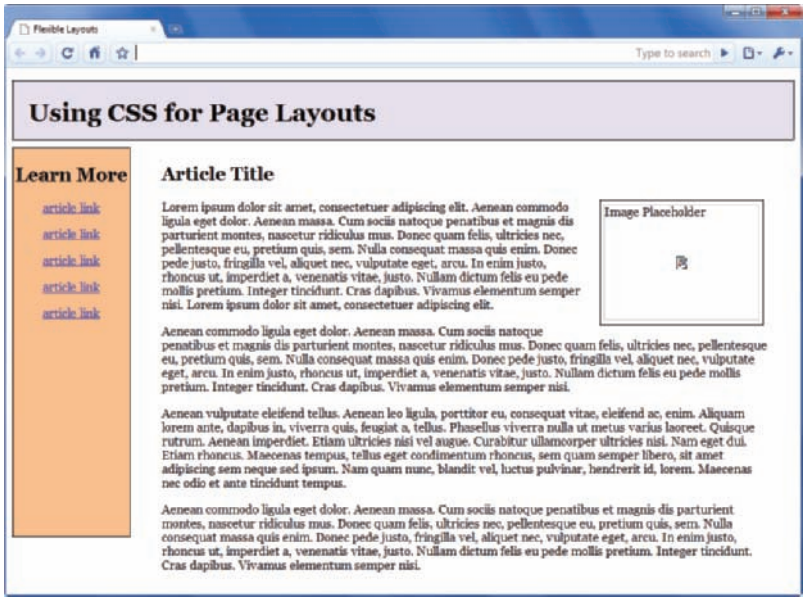
The header element has no width set, so it defaults to the width of the browser window as any normal flow element would. Below the header element are two floating elements, nav and article. These elements have flexible widths set as well. The style rules for these three elements follows:

```
#header {
   height: 75px;
   margin-top: 1em;
   margin-bottom: 10px;
   border: solid thin black;
   background-color: #e5dfec;
}

#nav {
   width: 15%;
   height: 500px;
   float: left;
   border: solid thin black;
   background-color: #fabf8f;
}

#article {
   width: 80%;
   height: 500px;
   float: left;
   margin-left: 2%;
   border: solid thin black;
   background-color: #fff;
}
```

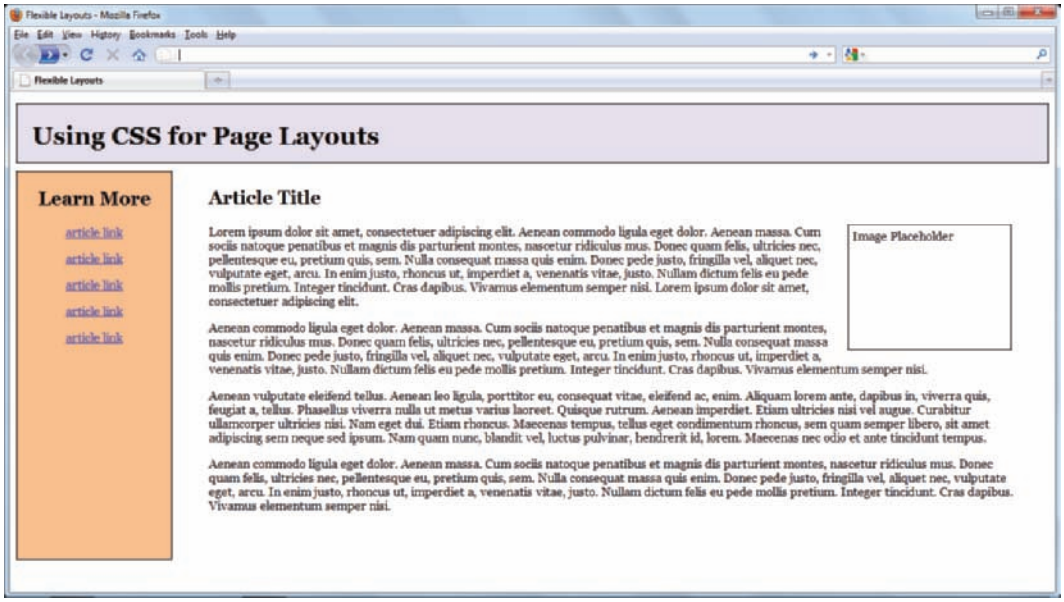
In the code example for Figure 7-14, note that the height property is used in both the nav and article elements. You normally want to avoid using height on elements that contain varying lengths of content, as too much content will overflow a fixed height.

Notice that the two widths for the floating elements do not equal 100% because the borders and margin contribute to the width of each element. Setting the values to equal 100% would overflow the layout window. The nav and article elements both float left. The article element has a 2% left margin, meaning that the margin changes size as the layout resizes to display a consistent gutter between the floating elements.

Flexible layouts offer the advantage of adapting to the user's browser size. From a design viewpoint, this can be less desirable because of the wide range of monitor sizes and resolutions. With a flexible layout, your content has to adapt and look good at wide range of layout sizes, which can be difficult to achieve. (Refer to the Broads Authority Web site in Figures 2-8 through 2-11 in Chapter 2 as an example.) Figure 7-15 shows a simple flexible layout at two widely different screen resolutions. Notice how the content adapts to the different resolutions. With a simple layout, flexible Web sites resize gracefully, but with more complex content and page designs, you may want to restrict your design with the min-width and max-width properties described in the next section.



1024 x 768 resolution



1366 x 768 resolution

Figure 7-15 Flexible layout adapts to different screen resolutions

Controlling Flexible Layout Width

You can control the compression and expansion of your content in a flexible layout by setting minimum and maximum widths for your content. As you saw in Chapter 6, the min-width and max-width properties let you set these properties for an element. You can add a wrapper element that contains the page elements, and then set the minimum and maximum sizes to control the boundaries of the page layout.

In the following style rule for the wrapper, the min-width value stops the shrinking of the layout at 750 pixels, which is an optimum width for an 800 x 600 monitor; the max-width value stops the expansion of the layout at 1220 pixels, the optimum width for a 1280 x 1024 display. Any monitor with a higher resolution will display the layout at 1220 pixels wide, maintaining the integrity of the layout.

```
div.wrapper {
   width: 100%;
   min-width: 750px;
   max-width: 1220px;
}
```

You would then apply the wrapper to a element that contains all other elements in your page layout. The following code shows a simplified version of what the wrapper looks like in HTML:

```
<div id="wrapper">  <!–opens wrapper -->

<div id="header">   header content...    </div>
<div id="nav">   nav content...    </div>
<div id="article">   article content...    </div>

</div>  <!–closes wrapper -->
```

You will build a wrapper division to contain a page layout like this in the Hands-on Activities later in the chapter.

To avoid guessing screen measurements, use a shareware tool such as Screen Ruler, available from *www.microfox.com*, to easily measure pixels on your computer monitor.

Activity: Creating a Flexible Layout

In the following steps, you will build a flexible two-column layout to practice using floating elements within a flexible design. As you work through the steps, refer to Figure 7-19 to see the results you will achieve. New code that you will add is shown in **blue**. Save your file and test your work in a browser as you complete each step.

To create the flexible layout:

1. Copy the **flexible_layout.htm** file from the Chapter07 folder provided with your Data Files to the Chapter07 folder in your work folder. (Create the Chapter07 folder, if necessary.)
2. Open the file **flexible_layout.htm** in your HTML editor, and save it in your work folder as **flexible_layout1.htm**.
3. In your browser, open the file **flexible_layout1.htm**. When you open the file, it looks like Figure 7-16.

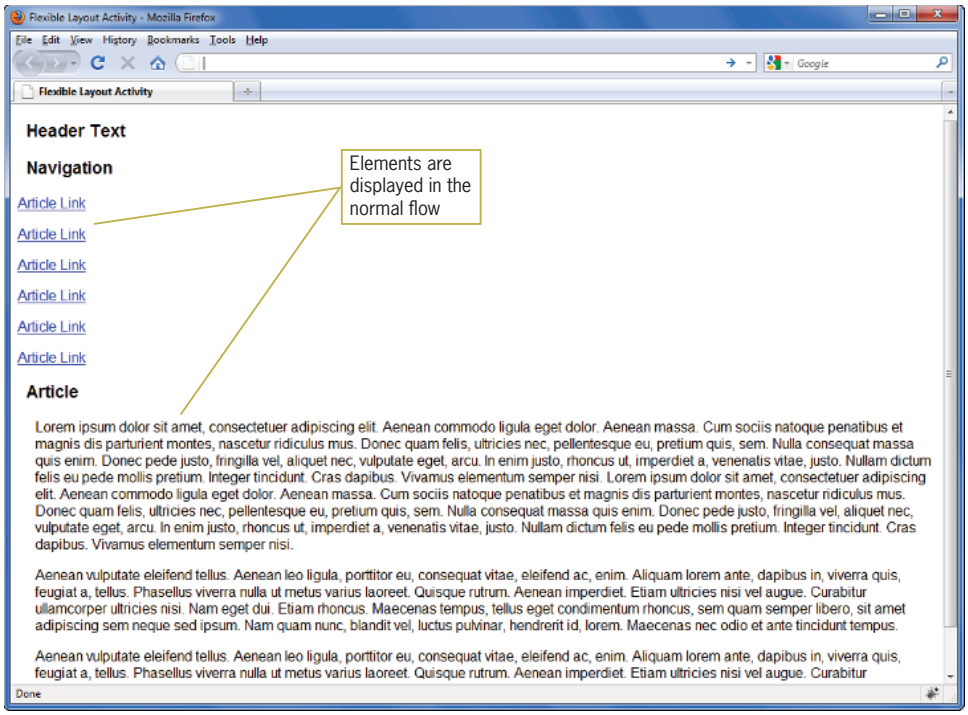


Figure 7-16 Flexible layout activity Web page

When you first look at the style rule you only see the basic formatting properties as shown:

```
<style type="text/css">
body {font-family: arial;}
h3 {font-family: arial; margin-left: 10px;}
.copy {margin-left: 20px}
</style>
```

There are no style rules for any of the content elements, so all are displayed in the normal flow, top to bottom in order and left-aligned to the browser window.

Each division element in the code already has the id names in place, as shown in the code for the header section:

```
<div id="header"><h3>Header Text</h3></div>
```

Formatting the Header

1. Begin by formatting the header division. Within the <style> element, add a style rule that selects the id named *header* and sets the height to 80 pixels.

```
#header {
height: 80px;
}
```

2. Add margins to offset the header from the top of the browser window and the content below by 10 pixels.

```
#header {
   height: 80px;
   margin-top: 10px;
   margin-bottom: 10px;
}
```

3. Align the text to the center of the header, add a solid thin black border, and apply light purple as the background color (#e5dfec).

```
#header {
   height: 80px;
   margin-top: 10px;
   margin-bottom: 10px;
   text-align: center;
   border: solid thin black;
   background-color: #e5dfec;
}
```

4. Save your file and preview the work in the browser. The header is now formatted, as shown in Figure 7-17.

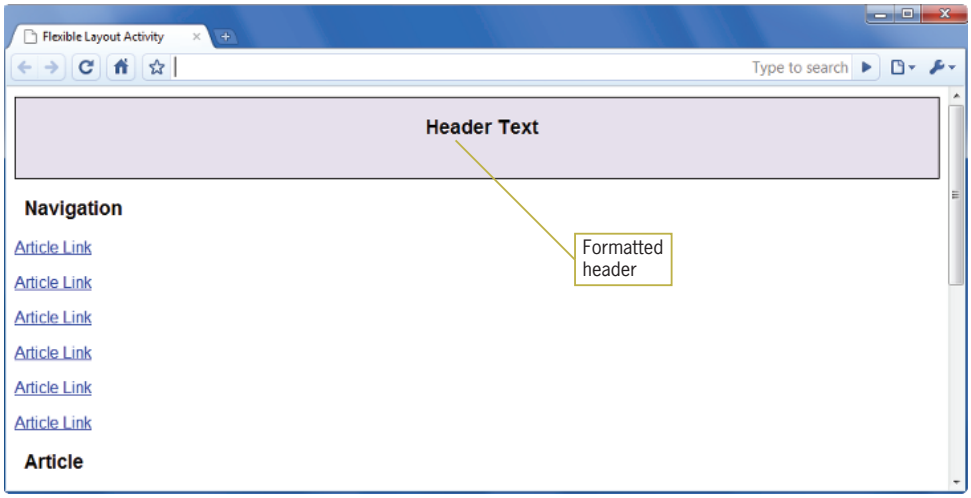


Figure 7-17 Completed header division

Formatting the Nav Division

1. Add a new selector for the id named *nav*, and set the division to float left with a width of 15%.

```
#nav {
   float: left;
   width: 15%;
}
```

2. Set the height to 500 pixels and align the text to center.

```
#nav {
   float: left;
   width: 15%;
   height: 500px;
   text-align: center;
}
```

3. Add a solid thin black border and set the background color to light orange (#fabf8f).

```
#nav {
   float: left;
   width: 15%;
   height: 500px;
   text-align: center;
   border: solid thin black;
   background-color: #fabf8f;
}
```

4. Save your file and preview the work in the browser. The nav division is now formatted, as shown in Figure 7-18. The article division needs some work, which you will do in the next set of steps.

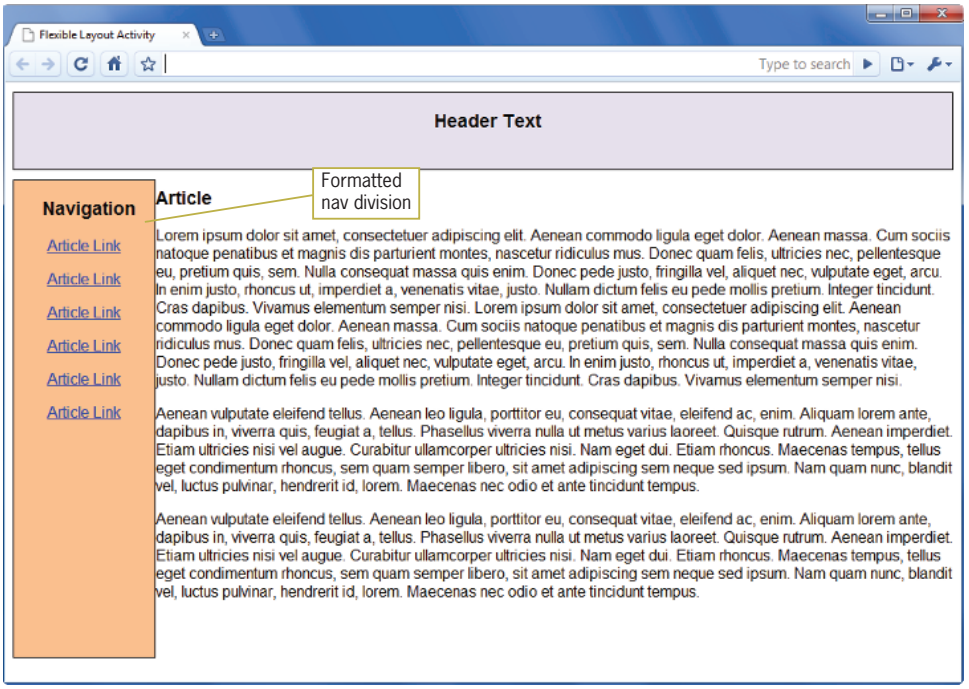


Figure 7-18 Completed nav division

Formatting the Article Division

1. Within the <style> element, add a style rule that selects the id named *article* and sets it to float left with a width of 80%.

```
#article {
   float: left;
   width: 80%;
}
```

2. Set a left margin of 2%. This allows the margin to adapt to different browser widths. Also set a background-color of white (#fff).

```
#article {
   float: left;
   width: 80%;
   margin-left: 2%;
   background-color: #fff;
}
```

3. Save your file and preview the work in the browser. The article division is now formatted, as shown in Figure 7-19, which completes the Web page layout. Notice as you shrink and expand the browser window the content adapts to fill the window.

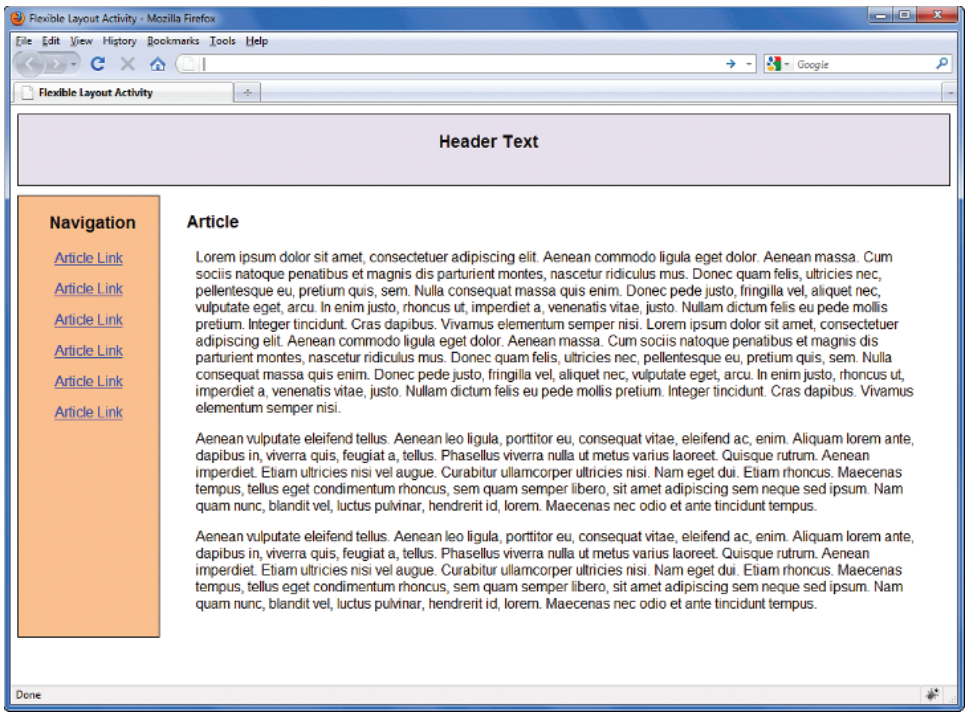


Figure 7-19 Completed flexible layout

The complete style sheet for the Web page in Figure 7-19 is as follows:

```
<style type="text/css">
body {font-family: arial;}
h3 {font-family: arial; margin-left: 10px;}
.copy {margin-left: 20px}

#header {
   height: 80px;
   margin-top: 10px;
   margin-bottom: 10px;
   text-align: center;
   border: solid thin black;
   background-color: #e5dfec;
}

#nav {
   float: left;
   width: 15%;
   height: 500px;
   text-align: center;
   border: solid thin black;
   background-color: #fabf8f;
}

#article {
   float: left;
```

```
    width: 80%;
    margin-left: 2%;
    background-color: #fff;
    }
</style>
```


Building a Fixed Page Layout

Fixed layouts remain constant despite the resizing of the browser in different screen resolutions and monitor sizes. Many designers prefer fixed layouts because they have more control over the finished design. They can also build more complex layouts because they can be fairly sure of consistent results. Fixed layouts are normally contained by a wrapper element that controls the page width and centers the page in the browser window. Within the wrapper, you can choose whether to contain only fixed-size elements, percentage elements, or a combination of the two. Because the outside width of the page is fixed, the design is more precise, and content can flow down the page as necessary. Pixel measurements are favored by many designers when creating fixed designs.

Refer to Figures 2-12 and 2-13 (The Boston Vegetarian Society Web site) for examples of a fixed layout.

Figure 7-20 shows a two-column layout that contains four elements.

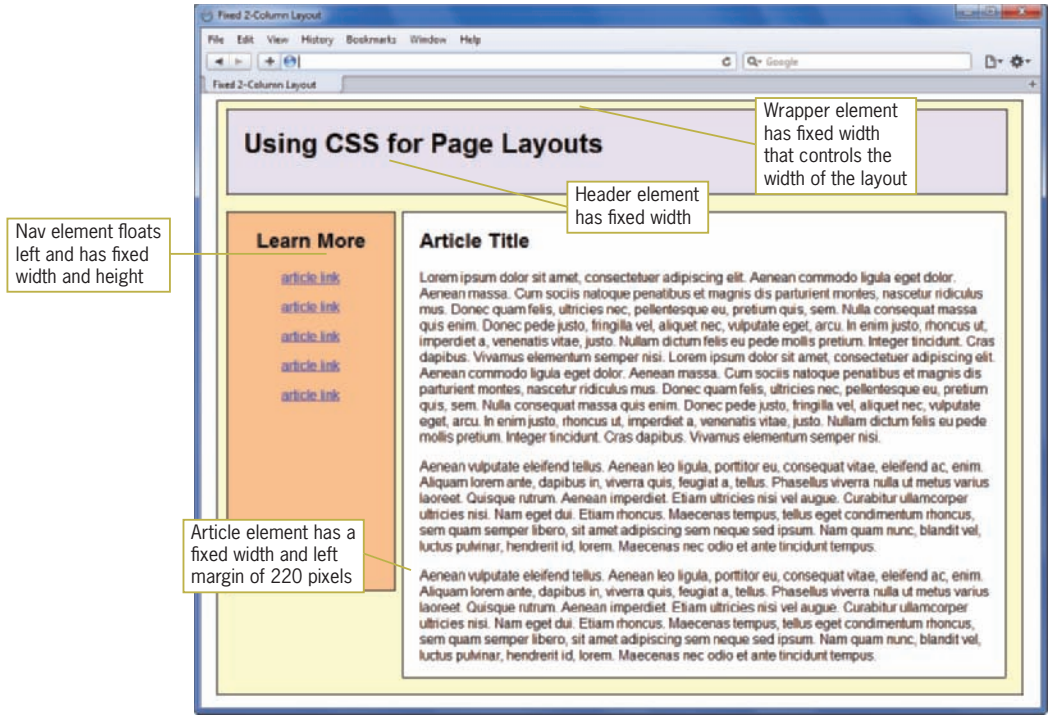


Figure 7-20 Two-column fixed layout

A wrapper division contains the other content elements and sets the fixed width for the layout. Header, nav, and article elements contain the page content. The nav element floats to the left and has a fixed width and height. The article element has a margin-left property that positions it on the page.

The style rules for these four elements follows:

```
#wrapper {
   width: 960px;
   margin-right: auto;
   margin-left: auto;
   border: thin solid black;
   background-color: #ffc;
}

#header {
   width: 930px;
   height: 100px;
   margin-top: 10px;
   margin-left: 10px;
   border: thin solid black;
   background-color: #e5dfec;
}

#nav {
   width: 200px;
   height: 450px;
   float: left;
   border: thin solid black;
   margin: 20px 20px 0px 10px;
   text-align: center;
   background-color: #fabf8f;
}

#article {
   width: 718px;
   border: thin solid black;
   margin: 20px 8px 20px 220px;
   background-color: #fff;
}
```

Notice that the wrapper element has a fixed width, but no fixed height, allowing content to flow down the page as necessary. The 960 pixel value for the width reflects the current base screen resolution of 1024 x 768; this may change based on your user's needs.

The header is in the normal flow and has a fixed width and height. The nav division floats left and has a fixed width and height as well. This element can be made more flexible by removing the height property and letting the content determine the height of the element.

The article element has a fixed width but no height, allowing the height of the element to be based on the amount of its content. The article element has a left margin (the last value in the margin property) that offsets the article 220 pixels from the left side of the browser window.

Controlling Fixed Layout Centering

Another benefit of using a wrapper division to contain your layout is the ability to automatically center the layout horizontally in the browser. This is a great solution for wide-screen monitors, as your layout will always be centered regardless of the screen resolution. Automatic centering is a simple use of the margin property. In the following style rule for the wrapper division in Figure 7-20, the margin-left and margin-right properties are set to *auto*, telling the browser to automatically proportion the extra space in the browser window, resulting in a centered layout.

```
#wrapper {
   width: 960px;
   margin-left: auto;
   margin-right: auto;
   border: thin solid black;
   background-color: #ffc;
}
```

Figure 7-21 shows the centered two-column fixed layout in a 1366 x 768 screen resolution.

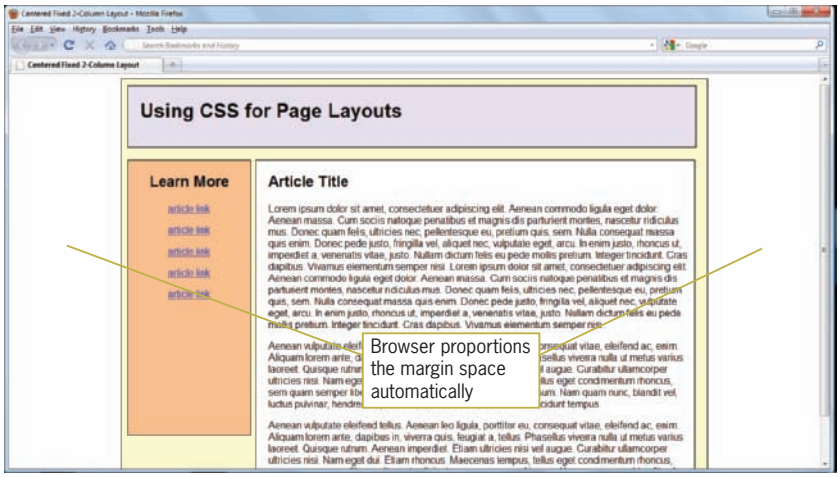


Figure 7-21 Centered two-column fixed layout

Activity: Creating a Fixed Layout

In the following steps, you will build a fixed three-column layout to practice using floating elements within a fixed design. As you work through the steps, refer to Figure 7-28 to see the results you will achieve. New code that you will add is shown in **blue**. Save your file and test your work in a browser as you complete each step.

To create the fixed layout:

1. Copy the **fixed_layout.htm** file from the Chapter07 folder provided with your Data Files to the Chapter07 folder in your work folder. (Create the Chapter07 folder, if necessary.)
2. Open the file **fixed_layout.htm** in your HTML editor, and save it in your work folder as **fixed_layout1.htm**.
3. In your browser, open the file **fixed_layout1.htm**. When you open the file it looks like Figure 7-22.

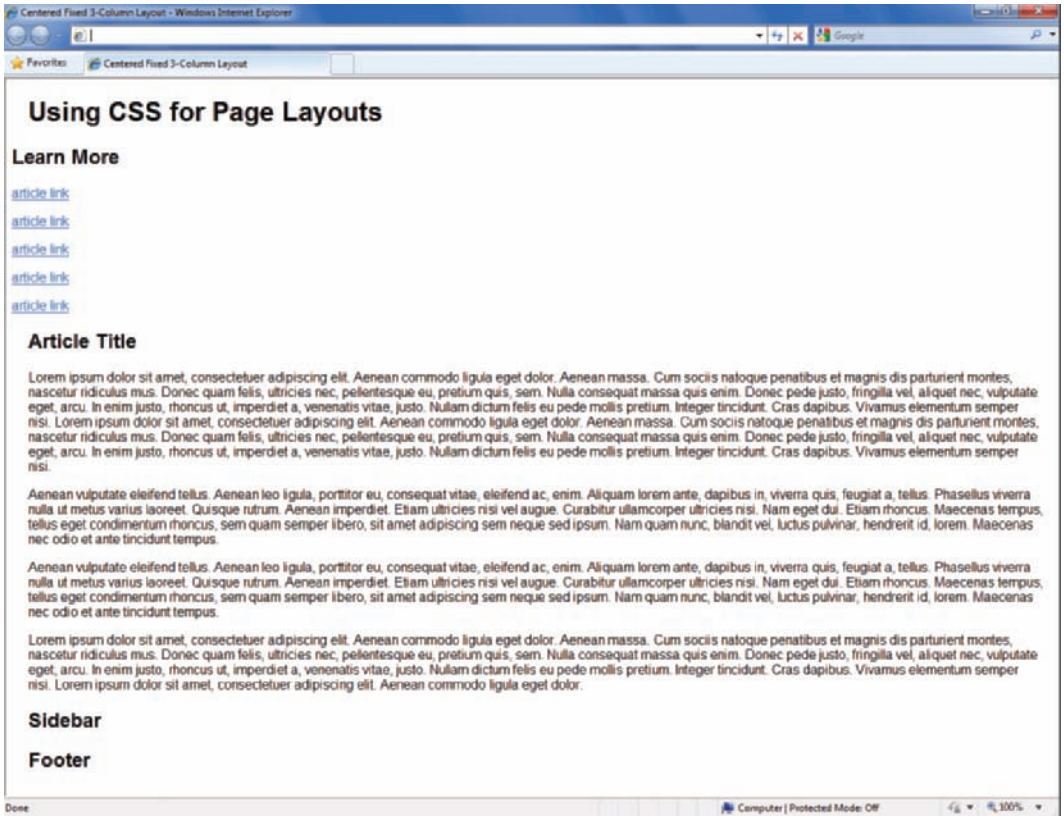


Figure 7-22 Fixed layout activity Web page

When you first look at the CSS style rules, you only see the basic formatting rules as shown:

```
<style type="text/css">
body {font-family: arial;}
h1 {margin-left: 20px;}
.copy {margin-left: 20px; margin-right: 10px;}
.title {margin-left: 20px;}
</style>
```

There are no style rules for any of the content elements, so all are displayed in the normal flow, top to bottom in order and left-aligned to the browser window.

Creating the Wrapper Division

1. Start the project by creating the wrapper division element that will contain the Web page content. Add a style rule that selects an id named *wrapper*. Set the width of the element to 1220 pixels for a 1280 x 1024 resolution.

```
#wrapper {
   width: 1220px;
}
```

2. Add margin-right and margin-left properties, and set these to *auto* so the page is automatically centered in the browser window.

```
#wrapper {
   width: 1220px;
   margin-right: auto;
   margin-left: auto;
}
```

3. Add a thin solid border and a background color of light yellow (#ffc) to complete the wrapper style rule.

```
#wrapper {
   width: 1220px;
   margin-right: auto;
   margin-left: auto;
   border: thin solid;
   background-color: #ffc;
}
```

4. Save your work and view it in the browser. Figure 7-23 shows the completed wrapper division.

If you are working with a monitor that has a lower resolution, such as 1024 x 768, you can adjust the width to a smaller width measurement, say 960 pixels. You will also have to adjust the widths of the contained division elements accordingly.

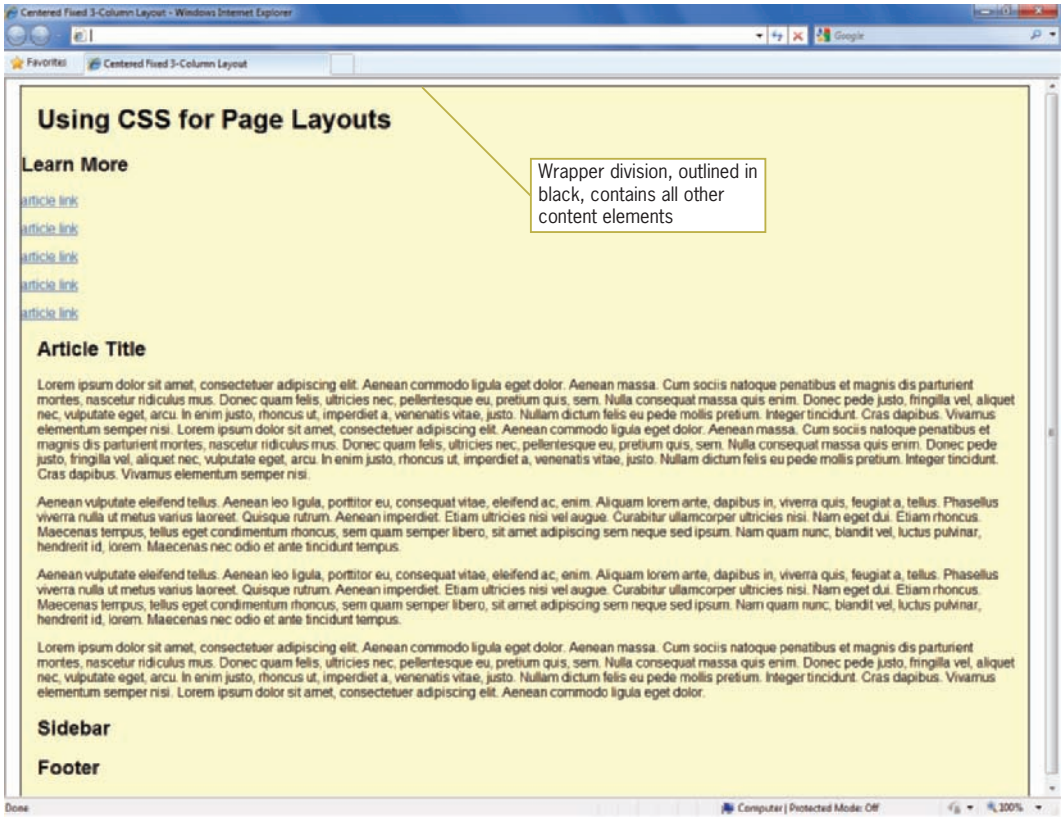


Figure 7-23 Completed wrapper division

Creating the Header Division

1. Add a style rule that selects an id named *header*. Set the width to 1200 pixels and the height to 100 pixels.

```
#header {
   width: 1200px;
   height: 100px;
}
```

2. Add a top margin of 10 pixels and a left margin of 10 pixels.

```
#header {
   width: 1200px;
   height: 100px;
   margin-top: 10px;
   margin-left: 10px;
}
```

3. Finish the header division by adding a solid thin border and a background color of light purple (#e5dfec).

```
#header {
   width: 1200px;
   height: 100px;
   margin-top: 10px;
   margin-left: 10px;
   border: thin solid;
   background-color: #e5dfec;
}
```

4. Save your work and view it in the browser. Figure 7-24 shows the completed header division.

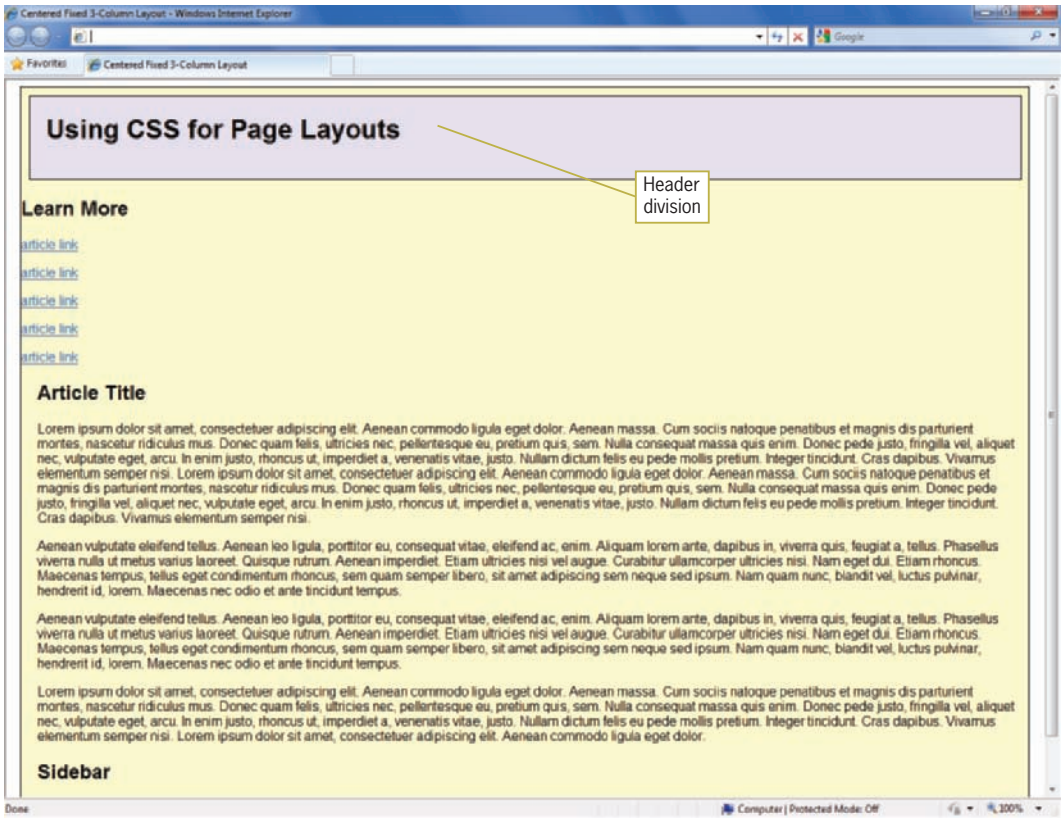


Figure 7-24 Completed header division

Creating the Nav Division

1. Add a style rule that selects an id named *nav*. Set the width to 200 pixels and the height to 600 pixels.

```
#nav {
   width: 200px;
   height: 600px;
}
```

2. Float the nav division to the left. Use the margin shorthand properties to set the following margin values:

   - Top margin: 20 pixels
   - Right margin: 14 pixels
   - Bottom margin: 0 pixels
   - Left margin: 10 pixels

```
#nav {
   width: 200px;
   height: 600px;
   float: left;
   margin: 20px 14px 0px 10px;
}
```

3. Complete the nav division by aligning the text to the center, adding a thin solid border, and setting the background color to light orange (#fabf8f).

```
#nav {
   width: 200px;
   height: 600px;
   float: left;
   margin: 20px 14px 0px 10px;
   text-align: center;
   border: thin solid;
   background-color: #fabf8f;
}
```

4. Save your work and view it in the browser. Figure 7-25 shows the completed nav division.

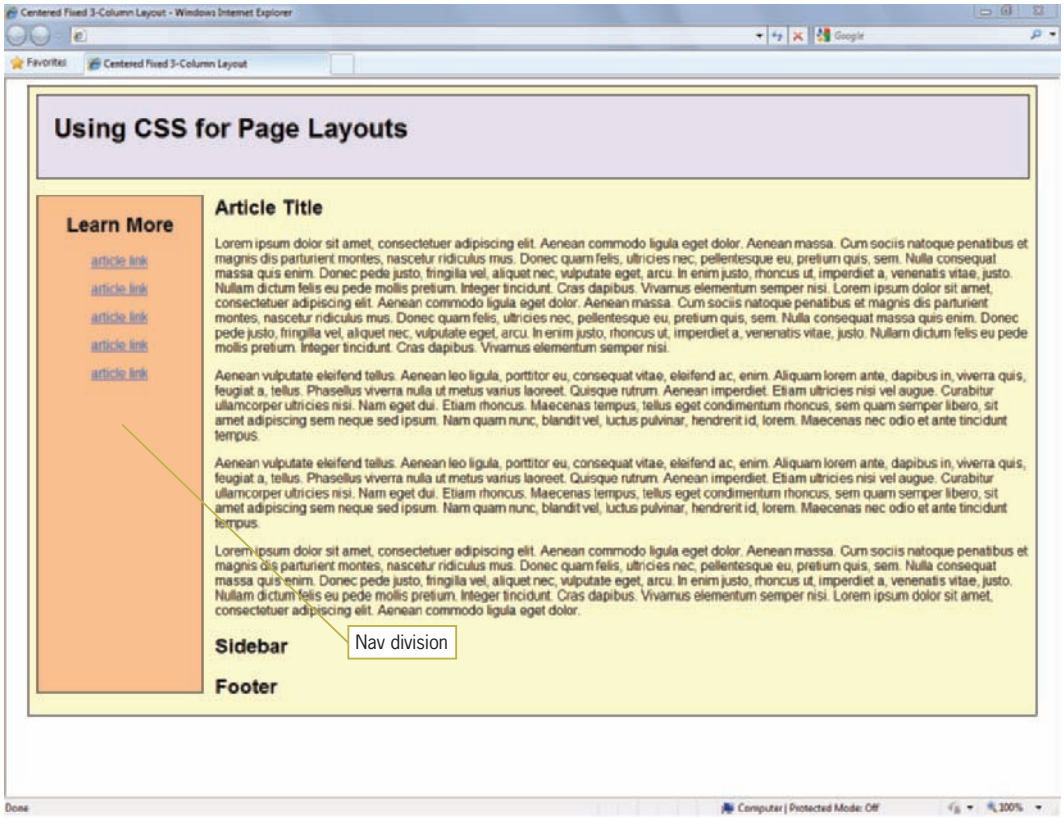


Figure 7-25 Complete nav division

Creating the Article Division

1. Add a style rule that selects an id named *article*. Set the width to 778 pixels and float the division to the left.

```
#article {
   width: 778px;
   float: left;
}
```

2. Use the margin shorthand properties to set the following margin values:
   - Top margin: 20 pixels
   - Right margin: 0 pixels

- Bottom margin: 20 pixels
- Left margin: 0 pixels

```
#article {
   width: 778px;
   float: left;
   margin: 20px 0px 20px 0px;
}
```

3. Add a thin solid border and a background color of white (#fff) to complete the article division.

```
#article {
   width: 778px;
   float: left;
   margin: 20px 0px 20px 0px;
   border: thin solid;
   background-color: #fff;
}
```

4. Save your work and view it in the browser. Figure 7-26 shows the completed article division.

   Notice that the floating elements are no longer properly contained within the wrapper division. You will fix this when you create the footer division at the end of this activity.

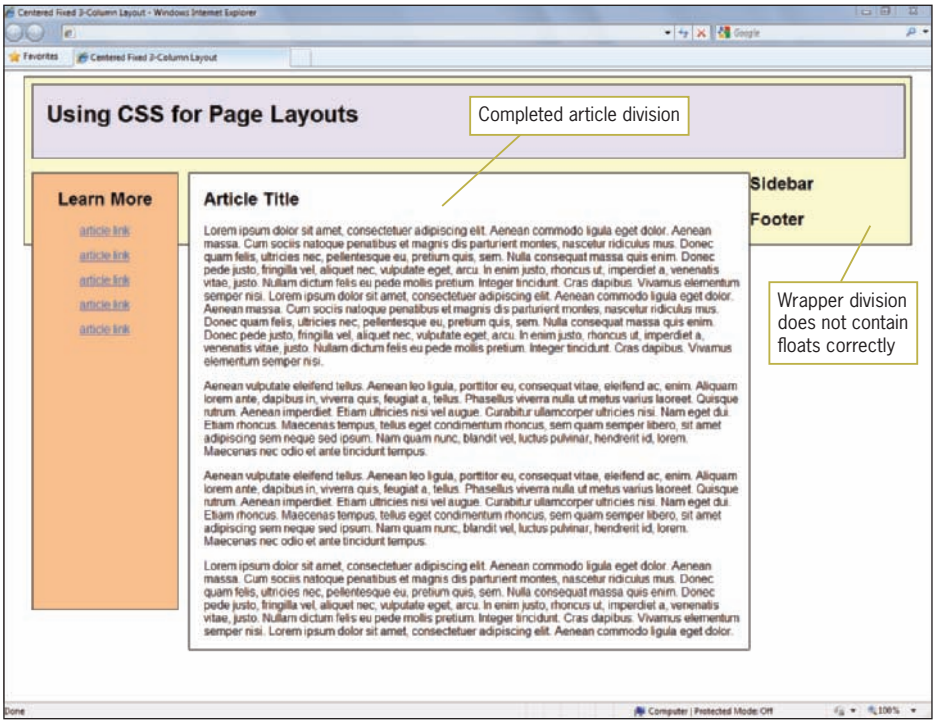


Figure 7-26 Completed article division

Creating the Sidebar Division

1. Add a style rule that selects an id named *sidebar*. Set the width to 200 pixels and the height to 600 pixels.

```
#sidebar {
   width: 200px;
   height: 600px;
}
```

2. Float the sidebar division to the right.

```
#sidebar {
   width: 200px;
   height: 600px;
   float: right;
}
```

3. Use the margin shorthand properties to set the following margin values:

   - Top margin: 20 pixels
   - Right margin: 8 pixels
   - Bottom margin: 0 pixels
   - Left margin: 0 pixels

```
#sidebar {
   width: 200px;
   height: 600px;
   float: right;
   margin: 20px 8px 0px 0px;
}
```

4. Add a thin solid border and a background color of light blue (#c6d9f1) to complete the sidebar division.

```
#sidebar {
   width: 200px;
   height: 600px;
   float: right;
   margin: 20px 8px 0px 0px;
   border: solid thin;
   background-color: #c6d9f1;
}
```

5. Save your work and view it in the browser. Figure 7-27 shows the completed sidebar division. The floating elements are still not properly contained within the wrapper division.

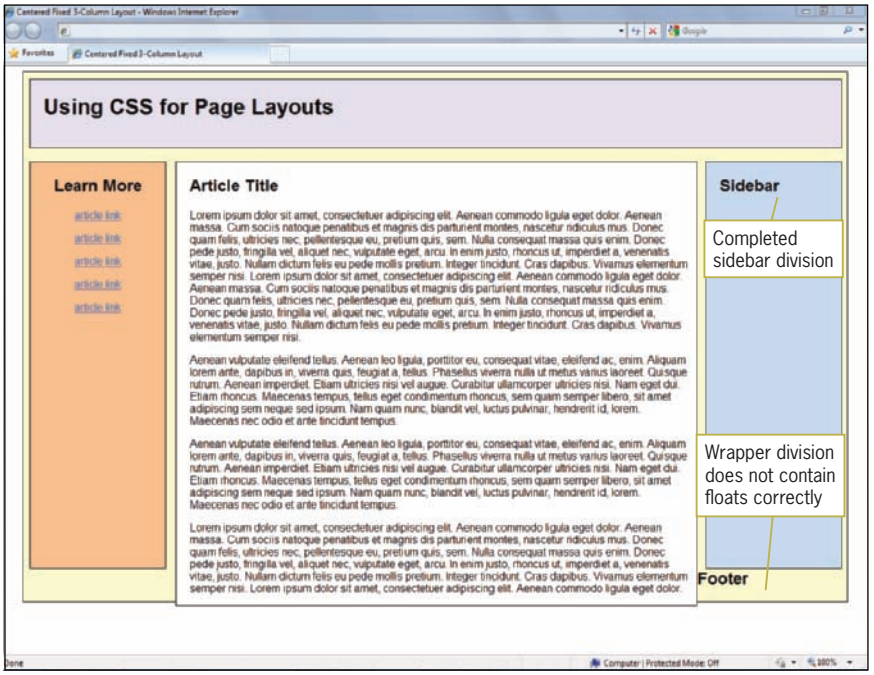


Figure 7-27 Completed sidebar division

Creating the Footer Division and Containing the Floats

1. Add a style rule that selects an id named *footer*. Set the width to 1200 pixels and the height to 75 pixels.

```
#footer {
   width: 1200px;
   height: 75px;
}
```

2. Add the clear property and set the value to *both*. This will fix the float containment problem you have seen in the previous steps.

```
#footer {
   width: 1200px;
   height: 75px;
   clear: both;
}
```

3. Finish the footer by adding a solid thin border, setting the left and bottom margins to 10 pixels, and adding a background color of pink (#efdfec).

```
#footer {
   width: 1200px;
   height: 75px;
   clear: both;
   border: solid thin;
   margin-left: 10px;
   margin-bottom: 10px;
   background-color: #efdfec;
}
```

4. Save your work and test the file in the browser. You will see that the wrapper now contains all of the floating elements, as shown in Figure 7-28.

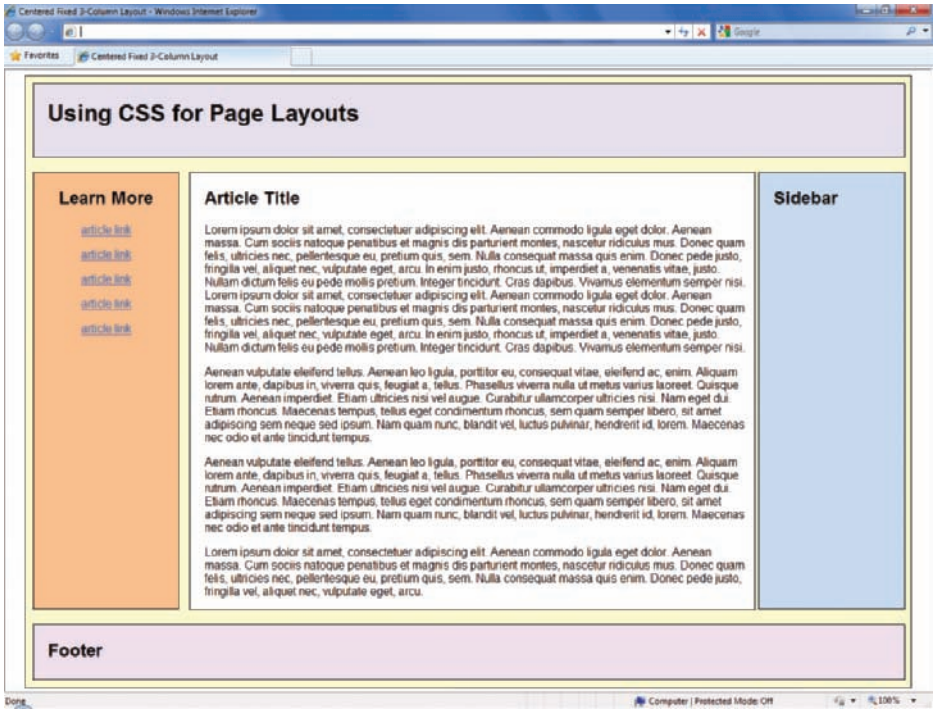


Figure 7-28 Completed fixed layout

Chapter Summary

In this chapter, you learned to apply the CSS box properties to Web page design. You learned about the normal flow of elements in the document and how the float property lets you remove elements from the normal flow. You learned how to create both flexible and fixed layouts, and to resolve problems such as dropped columns and incorrectly positioned floats. Finally, you applied what you learned by building two complete page layouts using the box properties.

- The normal flow dictates the way in which elements normally are displayed in the browser window.
- When you remove an element from the normal flow, you may see unexpected behavior from other elements that are following the normal flow.
- Remember to always use a width property for floating elements; otherwise, the element will extend across the page like elements in the normal flow.
- Remember to avoid using the height property unless you are containing elements such as images that do not change size.
- For fixed layouts, content elements are usually contained with a wrapper element that sets the width for the page layout.

Key Terms

column drop—A layout error that occurs when the total width of the columnar elements in a page layout exceeds the width of their containing element.

fixed layout—A layout that remains constant despite the resizing of the browser in different screen resolutions and monitor sizes.

flexible layout—A layout that shifts as the user resizes the window, wrapping text or adding white space as necessary to adapt to the size of the browser window. Also called liquid layouts.

float—To position an element by taking it out of the normal flow of the Web page layout.

normal flow—The sequence of element display in standard HTML from top to bottom and left to right until all elements that make up the Web page have been displayed.

wrapper—A division element designed to contain an entire Web page.

Review Questions

1. What is the normal flow for block-level elements?
2. Which element is the primary tool for creating sections of content in a Web page?
3. What is the common name for the content container for a Web page?
4. Would you normally use the height property when designing Web pages? Why or why not?
5. What property would you use to add gutters between columns of text?
6. What are the two methods of properly containing floats on a Web page?
7. Can you float elements within floating elements?
8. What causes column drops?
9. What property can be used to help position floating elements properly on a Web page?
10. What are the three possible values of the clear property?
11. What is the benefit of flexible layouts?
12. What is the benefit of a fixed layout?
13. What are the properties and values you would use to control flexible layout width?
14. What are the properties and values you would use to automatically center a fixed layout?

Hands-On Projects

1. Create a fixed three-column Web page with a header and a footer as shown in Figure 7-29. This page is designed for a 1024 x 768 resolution.

   a. Copy the **fixed_columns.html** file from the Chapter07 folder provided with your Data Files to the Chapter07 folder in your work folder. (Create the Chapter07 folder, if necessary.)

   b. Open the file **fixed_columns.html** in your HTML editor, and save it in your work folder as **fixed_columns1.html**.

   c. In your browser, open the file **fixed_columns1.html** and examine the page layout.

   d. Your goal is to use the values shown in Figure 7-29 to create a finished page with a fixed three-column layout, a header, and a footer.

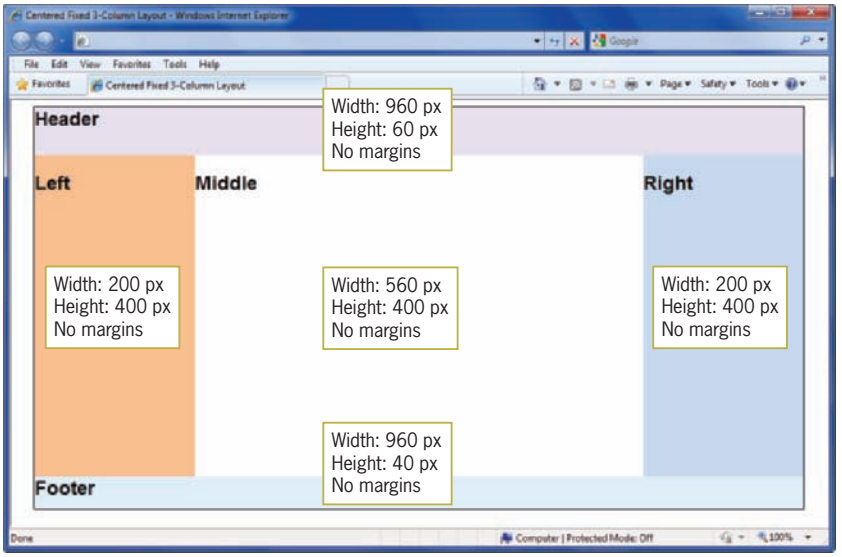


Figure 7-29 Fixed three-column Web page with a header and a footer

2. Fix the column drop shown in Figure 7-30. The completed Web page should look like Figure 7-31.

   a. Copy the **column_drop.html** file from the Chapter07 folder provided with your Data Files to the Chapter07 folder in your work folder. (Create the Chapter07 folder, if necessary.)

b. Open the file **column_drop.html** in your HTML editor, and save it in your work folder as **column_drop1.html**.

c. In your browser, open the file **column_drop1.html** and examine the page layout.

d. Your goal is to adjust the values used for the columns to create the Web page shown in Figure 7-31.

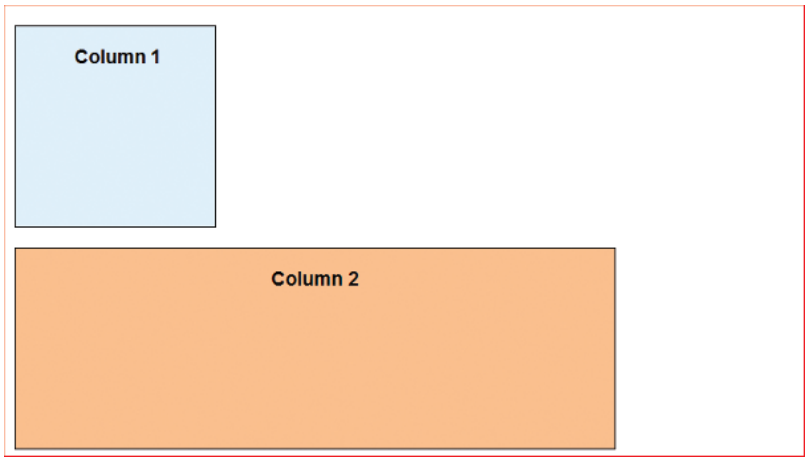


Figure 7-30 Column drop problem

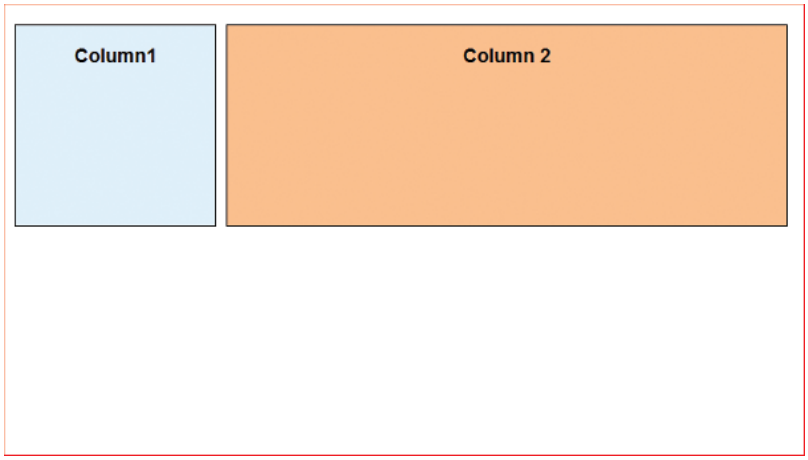


Figure 7-31 Column drop solution

3. Fix the float problem shown in Figure 7-32. The completed Web page should look like Figure 7-33.

   a. Copy the **float_problem.html** file from the Chapter07 folder provided with your Data Files to the Chapter07 folder in your work folder. (Create the Chapter07 folder, if necessary.)

   b. Open the file **float_problem.html** in your HTML editor, and save it in your work folder as **float_problem1.html**.

   c. In your browser, open the file **float_problem1.html** and examine the page layout.

   d. Your goal is to use floating and nonfloating elements to create the Web page shown in Figure 7-33.

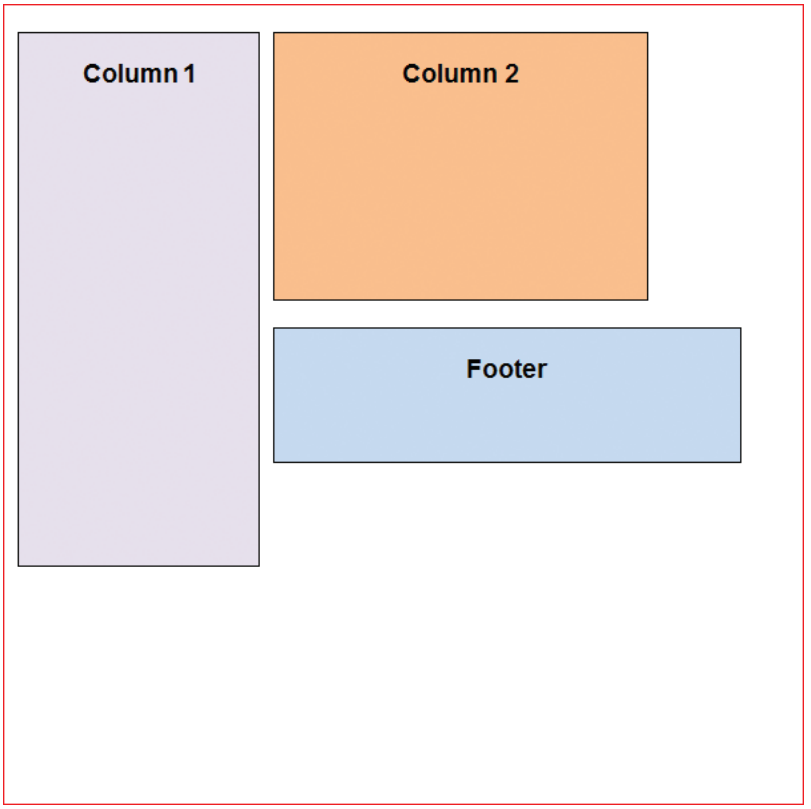


Figure 7-32 Float problem

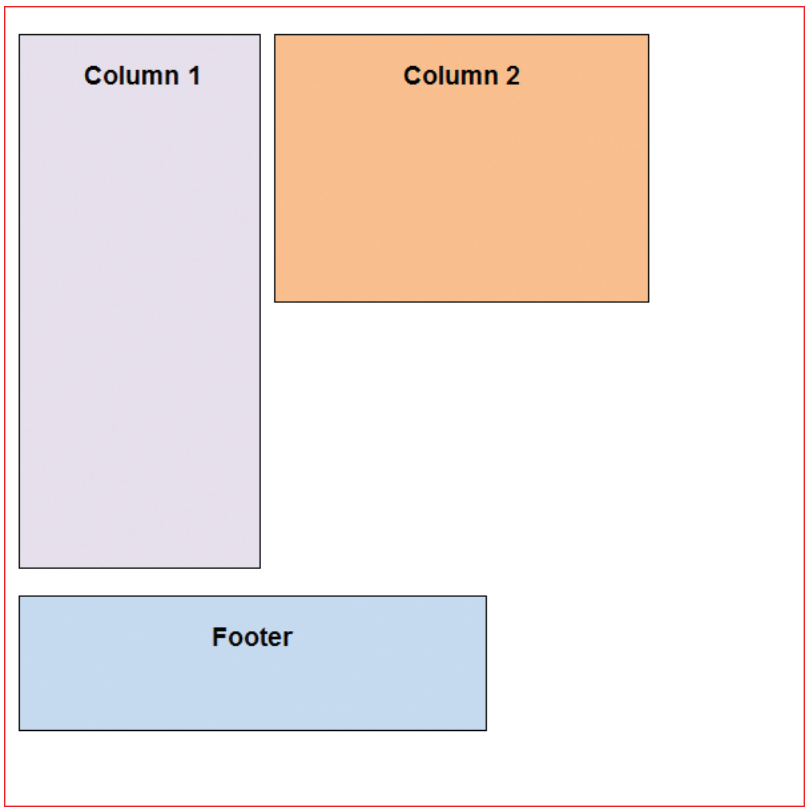


Figure 7-33 Float solution

Individual Case Project

Use your design sketches from Chapter 6 and start to build wireframe page mockups for the different page levels of your site. Decide on whether you are going to build a fixed or flexible design. Using the skills you learned in this chapter, build and submit page layouts for the different levels of information your site will contain. For example, you need to build a home page mockup, an article page mockup, and a section page mockup. Remember to test your page mockups with some text content and at different browser sizes and screen resolutions. Be prepared to explain your choices for your page layouts, such as why you chose fixed or flexible layouts.

Team Case Project

Each team member is responsible for different page templates for the different information levels of your Web site. Using the skills you learned in this chapter, build and submit page layouts for the different levels of information your site will contain. For example, one team member is responsible for the home page design, one for the section page design, and one for the article-level page design. Remember to test your page template using the text content at different browser sizes and screen resolutions. Be prepared to explain your choices for your page layouts, such as why you chose fixed or flexible layouts.