6

How to use CSS for page layout

In this chapter, you'll learn how to use CSS to control the layout of a page. That means that you can control where each of the HTML elements appear on the page. When you finish this chapter, you should be able to implement sophisticated 2- and 3-column page layouts.

How to float elements in 2- and 3-column layouts

To create a page layout with two or three columns, you usually float the elements that make up the columns of the page. You'll learn how to do that in the topics that follow.

How to float and clear elements

By default, the block elements defined in an HTML document flow from the top of the page to the bottom of the page, and inline elements flow from the left side of the block elements that contain them to the right side. When you *float* an element, though, it's taken out of the flow of the document. Because of that, any elements that follow the floated element flow into the space that's left by the floated element.

Figure 6-1 presents the basic skills for floating an element on a web page. To do that, you use the float property to specify whether you want the element floated to the left or to the right. You also have to set the width of the floated element. In the example, the aside is 150 pixels wide, and it is floated to the right. As a result, the section that follows flows into the space to the left of the aside.

Although you can use the float property with any block element, you can also use it with some inline elements. In chapter 4, for example, you learned how to float an image in the header of a document. When you float an img element, you don't have to set the width property because an image always has a default size.

By default, any content that follows a floated element in an HTML document will fill in the space to the side of the floated element. That includes block elements as well as inline elements.

However, if you want to stop the flow of elements into the space beside a floated element, you can use the clear property. In this example, this property is used to stop the footer from flowing into the space next to the aside. The value for this property can be left, right, or both, and either right or both will work if the element ahead of it is floated to the right. Similarly, left or both will work if the element ahead of it is floated to the left.

The properties for floating and clearing elements

Property	Description
`float`	A keyword that determines how an element is floated. Possible values are left, right, and none. None is the default.
`clear`	Determines whether an element is cleared from flowing into the space left by a floated element. Possible values are left, right, both, and none (the default).

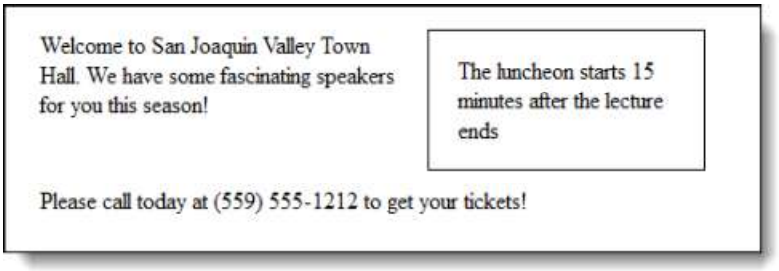
The HTML for a web page with a sidebar

```
<body>
    <aside>
        <p>The luncheon starts 15 minutes after the lecture ends</p>
    </aside>
    <section>
        <p>Welcome to San Joaquin Valley Town Hall. We have some fascinating
           speakers for you this season!</p>
    </section>
    <footer>
        <p>Please call today at (559) 555-1212 to get your tickets!</p>
    </footer>
</body>
```

The CSS for the web page for floating the sidebar

```
body { width: 500px; }
section, aside, footer {
    margin: 0;
    padding: 0px 20px; }
aside {
    margin: 0 20px 10px;
    width: 150px;
    float: right;
    border: 1px solid black; }
footer { clear: both; }
```

The web page in a browser



Description

- When you *float* an element to the right or left, the content that follows flows around it.
- When you use the float property for an element, you also need to set its width.
- To stop the floating before an element, use the clear property.
- In the example above, if the clear property for the footer isn't set, its content will flow into the space beside the floated element.

Figure 6-1 How to float and clear elements

How to use floating in a 2-column, fixed-width layout

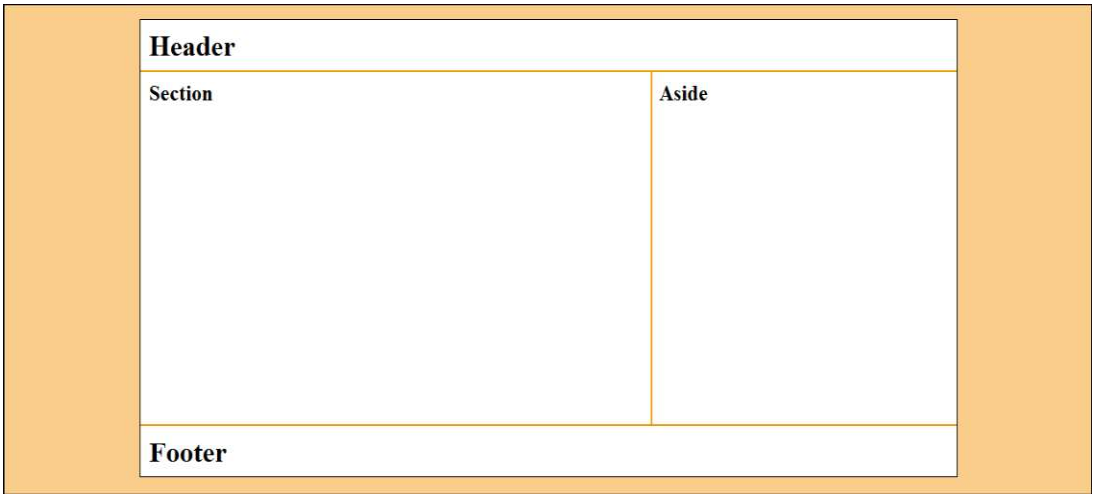
Figure 6-2 shows how floating can be used to create a 2-column, fixed-width page layout. Here, the HTML consists of four elements: header, section, aside, and footer.

In the CSS, you can see that the width is set for the body, section, and aside. Here, the width of the body must be the sum of the widths of the section and aside, plus the widths of any margins, padding, or borders for the section and aside. Since the right border for the section is 2 pixels and neither the section nor aside have margins or padding, the width of the body is 962 pixels (360 + 600 + 2).

After you set up the widths for the body and columns, you create the columns by floating the section to the left and the aside to the right. This will work whether the section or aside is coded first in the HTML.

Another alternative is to float both the section and aside to the left. But then, the section must come first in the HTML. In that case, though, the aside doesn't need to be floated at all. That's because the natural behavior of the aside is to flow into the space left by the floated section that's ahead of it in the HTML.

A 2-column web page with fixed-width columns



The HTML for the page

```
<body>
    <header><h1>Header</h1></header>
    <section><h1>Section</h1></section>
    <aside><h1>Aside</h1></aside>
    <footer><h1>Footer</h1></footer>
</body>
```

The CSS for the page

```
* { margin: 0; padding: 0; }
body {
    width: 962px;
    background-color: white;
    margin: 15px auto;
    border: 1px solid black; }
h1 { padding: 10px; }

header { border-bottom: 2px solid #ef9c00; }
section {
    height: 400px;     /* to give the sidebar some height for its border */
    width: 600px;
    float: left;
    border-right: 2px solid #ef9c00;
    float: left; }
aside {
    width: 360px;
    float: right; }
footer {
    clear: both;
    border-top: 2px solid #ef9c00; }
```

Description

- The section is floated to the left and the aside is floated to the right. Then, it doesn't matter whether the aside comes before or after the section in the HTML.
- Another alternative is to float both the section and the aside to the left, but then the section has to be coded before the aside in the HTML.

Figure 6-2 How to use floating in a 2-column, fixed-width layout

How to use floating in a 2-column, liquid layout

Instead of creating a *fixed layout* like the one in figure 6-2, you can create a *liquid layout*. With a liquid layout, the width of the page changes as the user changes the width of the browser window. Also, the width of one or more columns within the page changes.

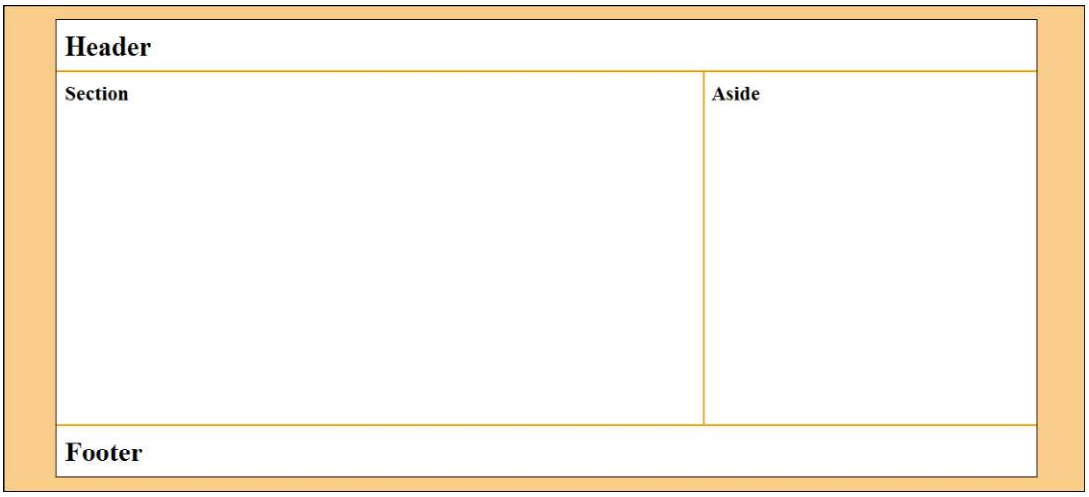
The key to creating a liquid layout is using percents to specify the widths of one or more elements. This is illustrated by figure 6-3. In both of these examples, the width of the page is set to 90%. That means that the page will always occupy 90% of the browser window, no matter how wide the window is. Of course, you can omit the width property entirely if you want the page to occupy 100% of the browser window.

In the first example, the widths of the section and aside are set to percents. This means that the widths of both columns will change if the user changes the width of the browser window. In this case, since the section has a 2-pixel right border, the sum of the widths of the section and aside is 99%, not 100%. However, if there wasn't a border, the sum could be 100%.

In the second example, the width of the aside is set to 360 pixels and no width is specified for the section. That way, if the user changes the width of the browser window, the width of the section will change so it occupies the width of the page minus the width of the aside. In this case, the border is applied to the left of the aside rather than to the right of the section because it's the width of the aside that's fixed.

As you decide whether to use a fixed or a liquid layout, your main consideration should be the content of the page. If the page consists of a lot of text, you probably won't want to use a liquid layout. That's because a page becomes more difficult to read as the length of a line of text gets longer. On the other hand, if you want the users to be able to use their browsers to increase the size of the text on a page, you might want to use a liquid layout. Then, the users can adjust the size of their browser windows to get the optimal line length for reading.

A 2-column web page with liquid widths for both columns



The CSS for the page when both columns are liquid

```
body {
    width: 90%;
    background-color: white;
    margin: 15px auto;
    border: 1px solid black; }
section {
    width: 66%;
    height: 400px;        /* to give this section some height */
    border-right: 2px solid #ef9c00;
    float: left; }
aside {
    width: 33%;
    float: right; }
```

The CSS for the page when the aside is fixed and the section is liquid

```
body {
    width: 90%;
    background-color: white;
    margin: 15px auto;
    border: 1px solid black; }
section {
    float: left; }
aside {
    height: 400px;        /* to give this aside some height */
    width: 360px;
    border-left: 2px solid #ef9c00;
    float: right; }
```

Description

- The benefit of using liquid column sizes is that the size of the page is adjusted to the resolution of the browser.
- The disadvantage is that changing the size of the columns may affect the typography or the appearance of the page.

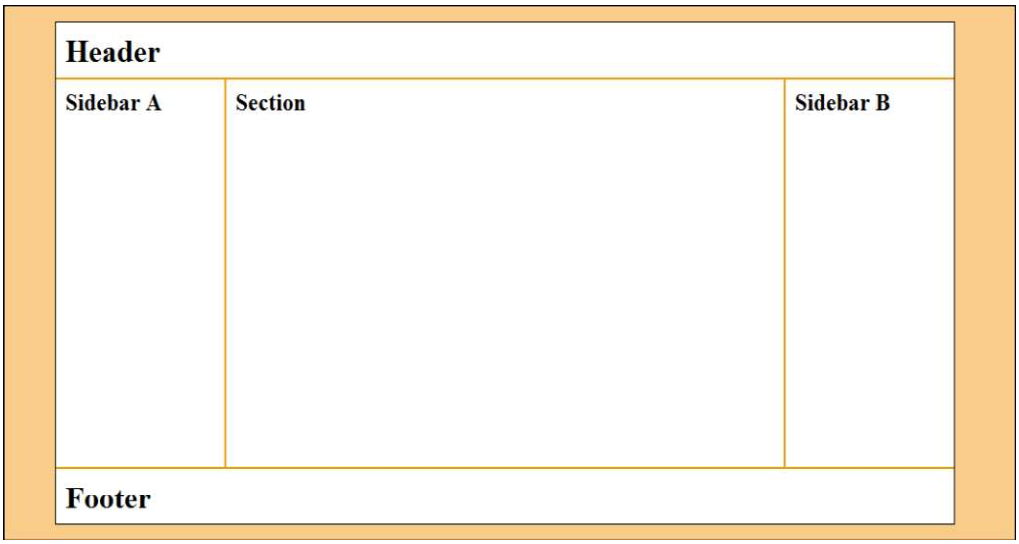
Figure 6-3 How to use floating in a 2-column liquid layout

How to use floating in a 3-column, fixed-width layout

Figure 6-4 shows how to take floating to one more level and thus create a 3-column, fixed-width page layout. Here, the width of the body is 964 pixels, which is the sum of the two sidebars, the section, and the two sidebar borders. Once those widths are set up, the first sidebar is floated to the left. The section is floated to the left. And the second sidebar is floated to the right.

One of the keys here is getting the widths right. If, for example, you set the body width to 960 pixels instead of 964 pixels, the two sidebars and the section won't fit into the width of the body. In that case, the section will flow beneath the left sidebar.

A 3-column web page with fixed-width columns



The HTML for the page

```
<body>
    <header><h1>Header</h1></header>
    <aside id="sidebarA"><h1>Sidebar A</h1></aside>
    <section><h1>Section</h1></section>
    <aside id="sidebarB"><h1>Sidebar B</h1></aside>
    <footer><h1>Footer</h1></footer>
</body>
```

The critical CSS for the page

```
body {
    width: 964px;
    background-color: white;
    margin: 15px auto;
    border: 1px solid black; }
#sidebarA {
    width: 180px;
    height: 400px;     /* to give the sidebar some height for its border */
    float: left;
    border-right: 2px solid #ef9c00; }
section {
    width: 600px;
    float: left; }
#sidebarB {
    width: 180px;
    height: 400px;     /* to give the sidebar some height for its border */
    float: right;
    border-left: 2px solid #ef9c00; }
```

Description

- The first aside is floated to the left; the section is floated to the left; and the second aside is floated to the right.
- You could get the same result by floating both asides and the section to the left.

Figure 6-4 How to use floating in a 3-column, fixed-width layout

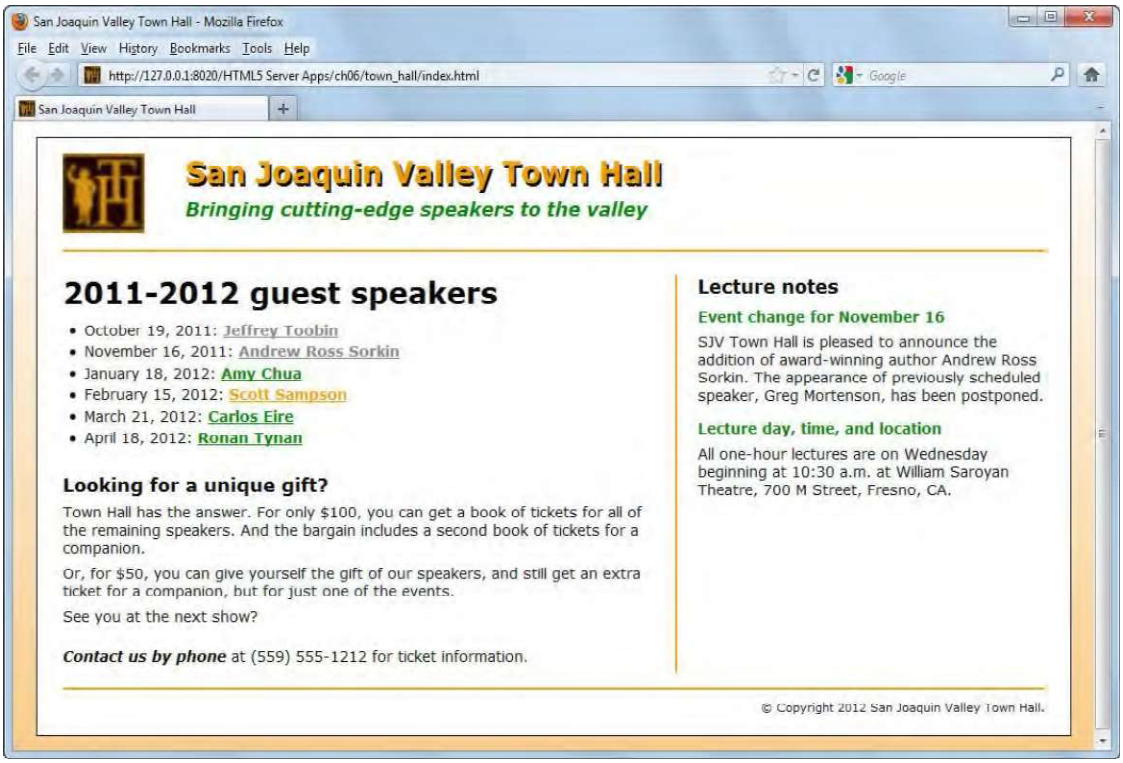
Two web pages that use a 2-column, fixed-width layout

To show how floating works in a more realistic application, the next topics present two pages of a web site along with their HTML and CSS.

The home page

Figure 6-5 shows the home page for the Town Hall web site. It uses a 2-column, fixed-width layout. Except for the aside, this is the same content that you saw in the page at the end of the last chapter. It's just formatted a little differently.

A home page with a sidebar floated to the right of a section



Description

- This web page illustrates a common page layout that includes a header, two columns, and a footer.
- The columns for this page are coded as a section element and an aside element.
- The columns are created by floating the aside element to the right so the section element that follows it in the HTML flows to its left.
- The image in the heading is floated to the left so it appears to the left of the h1 and h2 elements that follow it.
- A bottom border is applied to the header, and a top border is applied to the footer.
- A right border is applied to the section, not the aside, because the section content is longer than the aside content.
- If a left border was applied to the aside element, it would only be as long as the content in the aside.

Figure 6-5 A 2-column, fixed-width home page

The HTML for the home page

Figure 6-6 presents the HTML for this web page. Here, the only new content is in the aside. Note that the aside is coded before the section. Then, if the aside is floated to the right, the section will flow beside it.

Before looking at the CSS, take a minute to note the use of the HTML5 semantic elements. The header and footer start and end the HTML. The aside and section provide the content for the two columns. And the nav element within the section marks the navigation list. That helps make this code easy to understand.

The HTML for the home page (index.html)

```
<head>
    .
    <link rel="stylesheet" href="styles/main.css">
</head>
<body>
    <header>
        <img src="images/logo.jpg" alt="Town Hall Logo" width="80">
        <hgroup>
            <h1>San Joaquin Valley Town Hall</h1>
            <h2>Bringing cutting-edge speakers to the valley</h2>
        </hgroup>
    </header>
    <aside>
        <h1>Lecture notes</h1>
        <h2>Event change for November 16</h2>
        <p>SJV Town Hall is pleased to announce the addition of award-winning
           author Andrew Ross Sorkin. The appearance of previously scheduled
           speaker, Greg Mortenson, has been postponed.</p>
        <h2>Lecture day, time, and location</h2>
        <p>All one-hour lectures are on Wednesday beginning at 10:30 a.m. at
           William Saroyan Theatre, 700 M Street, Fresno, CA.</p>
    </aside>
    <section>
        <h1>2011-2012 guest speakers</h1>
        <nav>
            <ul>
                <li>October 19, 2011: <a class="date_passed"
                    href="speakers/toobin.html">Jeffrey Toobin</a></li>
                .
                .
                <li>February 15, 2012: <a href="sampson.html">
                    Scott Sampson</a></li>
                <li>March 21, 2012: <a href="speakers/eire.html">
                    Carlos Eire</a></li>
                <li>April 18, 2012: <a href="speakers/tynan.html">
                    Ronan Tynan</a></li>
            </ul>
        </nav>
        <h2>Looking for a unique gift?</h2>
        <p>Town Hall has the answer. For only $100, you can get
           a book of tickets for all of the remaining speakers. And the
           bargain includes a second book of tickets for a companion.</p>
        <p>Or, for $50, you can give yourself the gift of our speakers, and
           still get an extra ticket for a companion, but for just one of the
           events.</p>
        <p>See you at the next show?</p>
        <p id="contact_us"><em>Contact us by phone</em> at (559) 555-1212 for
           ticket information.</p>
    </section>
    <footer>
        <p>&copy; Copyright 2012 San Joaquin Valley Town Hall.</p>
    </footer>
</body>
</html>
```

Figure 6-6 The HTML for the home page

The CSS for the home page

Figure 6-7 presents the CSS for this web page. Since you've already seen most of this code in the earlier figures, you shouldn't have much trouble following it. But here are a few highlights.

First, the width of the body is set to 962 pixels, but the width of the section is set to 580 pixels and the width of the aside is set to 340 pixels, or a total of 920 pixels. The other 42 pixels come from right padding for the section (20 pixels), the left padding for the aside (20 pixels), and the 2 pixels for the right section border.

Second, the left margins of the h1 and h2 elements in the header are set to 120 pixels instead of being centered as they were in previous examples. This sets them to the right of the image in the header, which has been floated to the left.

Third, both the aside and section are floated to the right, but you could float the section to the left. This would also work if you didn't float the section at all since the aside comes first in the HTML.

The CSS for the home page (main.css)

```
section, aside, h1, h2, p, ul {
    margin: 0;
    padding: 0; }
section, aside {
    margin-top: 1.5em;
    margin-bottom: 1em; }

/* the changes to the body element in figure 5-14 */
body {
    width: 962px;
    /* no border radius or box shadow */
}

/* the changes to the header styles in figure 5-14 */
header h1 {
    color: #ef9c00;
    text-shadow: 2px 3px 0 black;
    margin-left: 120px;
    margin-bottom: .25em; }
header h2 {
    color: green;
    font-style: italic;
    margin-left: 120px; }
header img { float: left; }

/* the styles for the section */
section {
    width: 580px;
    border-right: 2px solid #ef9c00;
    padding-right: 20px;
    float: right; }
section h1 { margin-bottom: .35em; }
section h2 { margin-bottom: .35em; }
#contact_us { margin-top: 1em; }
a.date_passed  { color: gray; }

/* the styles for the sidebar */
aside {
    width: 340px;
    float: right;
    padding-left: 20px; }
aside h1 {
    font-size: 125%;
    padding-bottom: .5em; }
aside h2 {
    font-size: 100%;
    color: green;
    padding-bottom: .5em; }
aside p {
    margin-bottom: .5em; }

/* the styles for the footer that have been added or changed */
footer {
    clear: both;
    margin-top: 1em;
    border-top: 2px solid #ef9c00; }
```

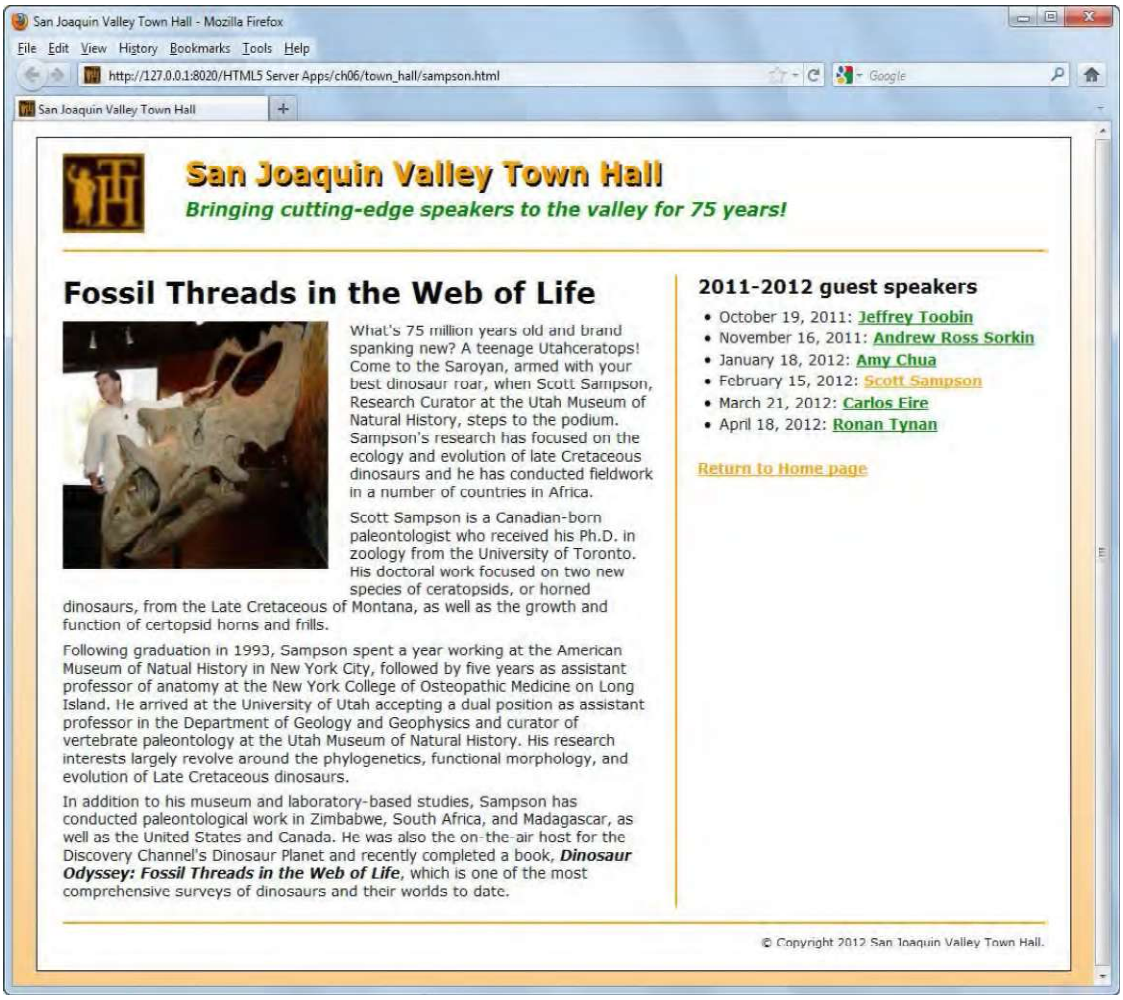
Figure 6-7 The CSS for the home page

The speaker page

Figure 6-8 shows the page that's displayed if you click on the fourth link of the home page. This page has the same header and footer as the home page, and it has a 2-column, fixed-width page layout like the one for the home page.

Unlike the home page, though, this page uses an article for its main content instead of a section. This is consistent with HTML5 semantics since the content is an article about the speaker. Within the article, an image of the speaker is floated to the left of the text.

A speaker page with a sidebar floated to the right of an article



Description

- The columns for this page are coded as an article element and an aside element.
- The columns are created by floating the aside element to the right so the article element that follows it in the HTML flows to its left.
- The image in the article is floated to the left. In the HTML, the image is coded after the h1 element and before the <p> elements that make up the article.
- A right border is applied to the article, not the aside, because the article content is longer than the aside content.

Figure 6-8 A 2-column, fixed-width speaker page

The HTML for the speaker page

Figure 6-9 presents just the HTML changes for the speaker page. For instance, it doesn't include the header and footer elements because they are the same for the home page and the speaker pages.

To start, the link element refers to a different style sheet named speaker.css. This style sheet can be used for all of the speaker pages.

After that, you can see the contents for the aside, which includes a nav element. You can also see the contents for the article, which includes an h1 element, an img element, and several <p> elements.

The CSS for the speaker page

Figure 6-9 also presents the CSS changes for the speaker page. Otherwise, the CSS is the same as the CSS for the home page.

First, the margins and padding are set up for the article and the aside, just as they were for the section and aside of the home page. Then, the article is formatted just like the section of the home page. In addition, though, the img element within the article is floated to the left.

Like the home page, both the aside and article are floated to the right, but you could float the article to the left. This would also work if you didn't float the article at all.

Because the CSS for the home page (main.css) and the CSS for the speaker pages (speaker.css) are so much alike, you could combine the CSS into a single file. Often, though, it's easier to manage the CSS if you keep it in separate files. Then, if you want to modify the CSS for the speaker pages, you don't have to worry about affecting the home page.

The HTML changes for the speaker page (sampson.html)

```
<head>
    .
    <link rel="stylesheet" href="styles/speaker.css">
</head>
<aside>
    <h1>2011-2012 guest speakers</h1>
    <nav>
        <ul>
            <li>October 19, 2011: <a class="date_passed"
                href="speakers/toobin.html">Jeffrey Toobin</a></li>
            .
            .
            <li>April 18, 2012: <a href="speakers/tynan.html">
                Ronan Tynan</a></li>
        </ul>
        <p><a href="index.html">Return to Home page</a></p>
    </nav>
</aside>
<article>
    <h1>Fossil Threads in the Web of Life</h1>
    <img src="images/sampson_dinosaur.jpg" alt="Scott Sampson" width="260">
    <p>What's 75 million years old and brand spanking new? A teenage
       ...
       ...</p>
    <p>Scott Sampson is a Canadian-born paleontologist who received his
       ...
       ...</p>
    <p>Following graduation in 1993, Sampson spent a year working at the
       ...
       ...</p>
    <p>In addition to his museum and laboratory-based studies, Sampson has
       ...
       ...</p>
</article>
```

The CSS changes for the speaker page (speaker.css)

```
article, aside, h1, h2, p, ul {
    margin: 0;
    padding: 0; }
article, aside {
    margin-top: 1.5em;
    margin-bottom: 1em; }

/* the styles for the article */
article {
    width: 580px;
    border-right: 2px solid #ef9c00;
    padding-right: 20px;
    float: right; }
article h1 {
    margin-bottom: .35em; }
article img {
    float: left;
    margin: 0 1.5em 1em 0; }
```

Figure 6-9 The HTML and CSS for the speaker page

How to use CSS3 to create text columns

So far, we've been talking about the columns in a page layout. But CSS3 provides a new feature that makes it easy to create text columns. For instance, you can easily format an article into two or more text columns within a 2- or 3-column page layout.

The CSS3 properties for creating text columns

Figure 6-10 summarizes the properties for creating text columns. For instance, the column-count property automatically formats the text within an article into the number of columns specified. You can use the column-gap property to set the width of the gaps between columns. You can use the column-rule property to set borders within the columns. And you can use the column-span property to cause an element to span the columns.

Three of these properties are illustrated by the example in this figure. Here, an article is formatted into 3 columns with 25 pixel gaps, and 2 pixel rules between the columns. To fit this into this figure, the bottom of the article is truncated, but the entire article is formatted correctly.

The problem with this feature is that only Opera supports it in its native form, IE doesn't support it at all, and the other browsers require prefixes. That's why the code in this example uses the required prefixes. Nevertheless, this is a powerful feature that can easily improve the readability of an article by shortening the line length.

When using this feature, though, you don't want to make the columns too narrow because that can make the text more difficult to read. Also, you don't want to justify the text because that can cause gaps between the words.

The primary CSS3 properties for creating text columns

Property	Description
`column-count`	The number of columns that the text should be divided into.
`column-gap`	The width between the columns. Otherwise, this is set by default.
`column-rule`	Defines a border between columns.
`column-span`	With the value "all", this property can be used to have everything that precedes the element span all of the columns, but at present this is only supported by Chrome and Safari, not Opera.

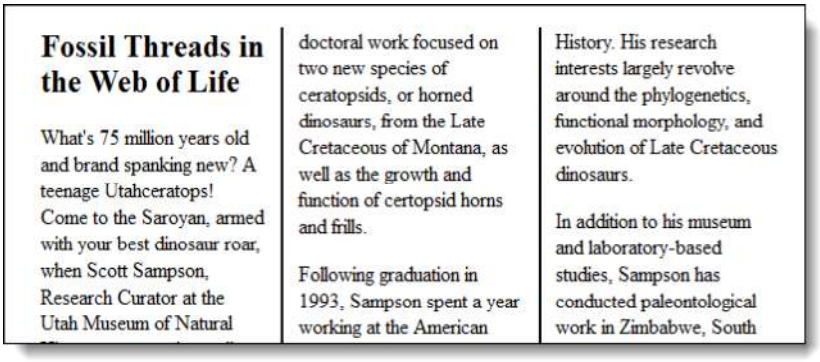
3 columns with default-sized gaps

```
article {
    -moz-column-count: 3;
    -webkit-column-count: 3;
    column-count: 3; }
```

3 columns with 25px gaps and 2px rules between the columns

```
article {
    -moz-column-count: 3;
    -webkit-column-count: 3;
    column-count: 3;
    -moz-column-gap: 25px;
    -webkit-column-gap: 25px;
    column-count-gap: 25px;
    -moz-column-rule: 2px solid black;
    -webkit-column-rule: 2px solid black;
    column-rule: 2px solid black; }
```

3 columns with 25px gaps and 2px rules in a browser window



Fossil Threads in the Web of Life

What's 75 million years old and brand spanking new? A teenage Utahceratops! Come to the Saroyan, armed with your best dinosaur roar, when Scott Sampson, Research Curator at the Utah Museum of Natural

doctoral work focused on two new species of ceratopsids, or horned dinosaurs, from the Late Cretaceous of Montana, as well as the growth and function of certopsid horns and frills.

Following graduation in 1993, Sampson spent a year working at the American

History. His research interests largely revolve around the phylogenetics, functional morphology, and evolution of Late Cretaceous dinosaurs.

In addition to his museum and laboratory-based studies, Sampson has conducted paleontological work in Zimbabwe, South

Description

- At this writing, only Opera supports these properties in their native form. Firefox requires the -moz- prefix, Chrome and Safari require the -webkit- prefix, and IE provides no support for these properties.
- If you want the heading of an article to span the columns, you can code the HTML for the heading before the article. Then, you don't have to use the column-span property.
- If the columns are too narrow, an article becomes harder to read.

Figure 6-10 The properties for creating text columns

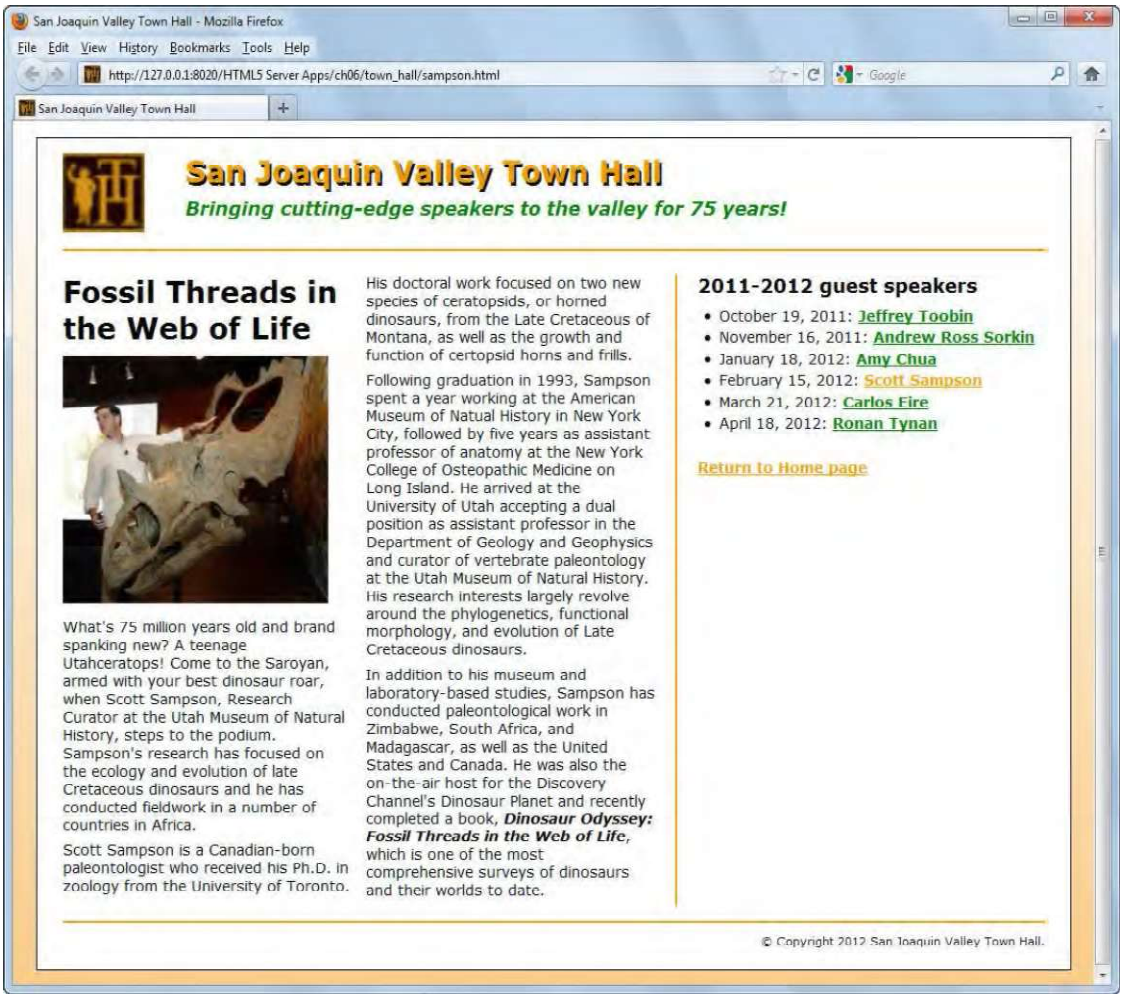
A 2-column web page with a 2-column article

Figure 6-11 shows how easily this feature can add columns to the article in the speaker page of figure 6-8. It just takes one column-count property, plus the two prefixed versions of it. Then, if a browser doesn't support this property, it just ignores it, so there's no harm done.

If you want the heading to span the columns, you have to do some other work because the column-span property is currently supported only by Safari and Chrome. But this is still an excellent feature that you can start using right away.

Incidentally, if you want to float the image that's coded at the start of the text to the right when you use two or more columns, it won't work the way you want. Instead of floating the image in the rightmost column, the browser will float the image in the first column. Although there are ways around this, it's usually best to float the image to the left when you use this feature.

A web page with a two-column article



The CSS for creating the columns

```
article {
    -moz-column-count: 2;
    -webkit-column-count: 2;
    column-count: 2;
}
```

Description

- This shows how easy it is to divide an article in columns.
- If you compare this page with the one in figure 6-8, you can see that this page is easier to read because the lines are shorter, but not too short.
- If a browser doesn't support this feature, the page is displayed as in figure 6-8, which is acceptable.
- If you want the heading in the article to span the columns without using the CSS3 column-span property, you have to code the heading before the article in the HTML.
- In general, it's best to float an image within the columns area to the left so it will be in the leftmost column. If you float it to the right, the browser will float the image to the right, but in the first column, which usually isn't what you want.

Figure 6-11 A 2-column web page with a 2-column article

How to position elements

Although floating provides an easy way to create pages with two or more columns, you may occasionally need to use other positioning techniques. You'll learn about those techniques in the topics that follow.

Four ways to position an element

To position the elements on a page, you use the properties shown in the first table in figure 6-12. The first property, position, determines the type of positioning that will be used. This property can have four different values as shown in the second table.

The first value, static, is the default. This causes elements to be placed in the normal flow.

The second value, absolute, removes the element from the normal flow and positions it based on the top, bottom, right, and left properties you specify. When you use *absolute positioning*, the element is positioned relative to the closest containing block. If no containing block is positioned, the element is positioned relative to the browser window.

The third value, fixed, works like absolute in that the position of the element is specified using the top, bottom, left, and right properties. Instead of being positioned relative to a containing block, however, an element that uses *fixed positioning* is positioned relative to the browser window. That means that the element doesn't move when you scroll through the window.

The fourth value, relative, causes an element to be positioned relative to its normal position in the flow. When you use the top, bottom, left, and right properties with *relative positioning*, they specify the element's offset from its normal position.

One more property you can use to position elements is z-index. This property is useful if an element that you position overlaps another element. In that case, you can use the z-index property to determine which element is on top.

Properties for positioning elements

Property	Description
`position`	A keyword that determines how an element is positioned. See the table below for possible values.
`top, bottom, left, right`	For absolute or fixed positioning, a relative or absolute value that specifies the top, bottom, left, or right position of an element's box. For relative positioning, the top, bottom, left, or right offset of an element's box.
`z-index`	An integer that determines the stack level of an element whose position property is set to absolute, relative, or fixed.

Possible values for the position property

Value	Description
`static`	The element is placed in the normal flow. This is the default.
`absolute`	The element is removed from the flow and is positioned relative to the closest containing block that is also positioned. The position is determined by the top, bottom, left, and right properties.
`fixed`	The element is positioned absolutely relative to the browser window. The position is determined by the top, bottom, left, and right properties.
`relative`	The element is positioned relative to its position in the normal flow. The position is determined by the top, bottom, left, and right properties.

Description

- By default, static positioning is used to position block elements from top to bottom and inline elements from left to right.
- To change the positioning of an element, you can code the position property. In most cases, you also code one or more of the top, bottom, left, and right properties.
- When you use absolute, relative, or fixed positioning for an element, the element can overlap other elements. Then, you can use the z-index property to specify a value that determines the level at which the element is displayed. An element with a higher z-index value is displayed on top of an element with a lower z-index value.

Figure 6-12 Four ways to position an element

How to use absolute positioning

To give you a better idea of how positioning works, figure 6-13 presents an example of absolute positioning. Here, the aside element is positioned absolutely within its containing element, which is the body of the HTML document.

For this to work, the containing element must also be positioned. However, the positioning can be either relative or absolute, and the top, bottom, left, and right properties don't have to be set. Since they aren't set in this example, the body is positioned where it is in the natural flow. In other words, nothing changes. However, because the body is positioned, the elements that it contains can be absolutely positioned within it. In this case, the aside element is positioned 30 pixels from the right side of the body and 50 pixels from the top.

When you use absolute positioning, you typically specify the top or bottom and left or right properties. You also are likely to specify the width or height of the element. However, you can also specify all four of the top, bottom, left, and right properties. Then, the height and width are determined by the difference between the top and bottom and left and right properties.

How to use fixed positioning

Fixed positioning is like absolute positioning except that it's positioning is relative to the browser window. So, to change the example in figure 6-13 from absolute positioning to fixed positioning requires just two changes. First, you delete the position property for the body since fixed positioning is relative to the browser window. Second, you change the position property for the aside to fixed.

The benefit of fixed positioning is that the element that's fixed stays there when you scroll down the page. If, for example, the section in this figure was so long that it required scrolling, the aside would stay where it is while the user scrolls.

The HTML for a web page

```
<body>
    <section>
        <h1>Our speakers for 2011-2012</h1>
        <ul>
            <li>October 19, 2011: <a href="speakers/toobin.html">
                Jeffrey Toobin</a></li>
            <li>November 16, 2011: <a href="speakers/sorkin.html">
                Andrew Ross Sorkin</a></li>
            <li>January 18, 2012: <a href="speakers/chua.html">
                Amy Chua</a></li>
        </ul>
        <p>Please contact us for tickets.</p>
    </section>
    <aside>
        <p><a href="raffle.html">Enter to win a free ticket!</a></p>
    </aside>
</body>
```

The CSS for the web page with absolute positioning

```
p { margin: 0; }
body {
    width: 500px;
    margin: 0 25px 20px;
    border: 1px solid black;
    position: relative; }  /* not needed for fixed positioning */
aside {
    width: 80px;
    padding: 1em;
    border: 1px solid black;
    position: absolute;    /* change to fixed for fixed positioning */
    right: 30px;
    top: 50px; }
```

The web page with absolute positioning in a browser

Description

- When you use *absolute positioning*, the remaining elements on the page are positioned as if the element weren't there. As a result, you may need to make room for the positioned element by setting the margins or padding for other elements.
- When you use *fixed positioning* for an element, the positioning applies to the browser window and the element doesn't move even when you scroll.

Figure 6-13 How to use absolute and fixed positioning

A table of contents that uses positioning

Because floating works so well, you won't need to use positioning much. Occasionally, though, it comes in handy.

In figure 6-14, for example, you can see the table of contents for a book that makes good use of positioning. If positioning weren't used, you would have to use a table to display a table of contents like this, which would take more code and be more difficult.

In the HTML for the table of contents, a section element is used as the container for the table of contents. Within this container, h2 elements are used for the sections of the table of contents and h3 elements are used for the chapters. Within the h2 and h3 elements, span elements with "title" as the class name are used to specify the titles of the sections and chapters. And within the h3 elements, span elements with "number" as the class name are used to specify the page numbers of the chapters.

In the CSS for the table of contents, you can see that the h2 and h3 elements are given relative positioning. However, the top, bottom, left, and right properties have been omitted. Because of that, the elements aren't actually offset, but they can be used as containers for absolute positioning. Then, the title and number classes are positioned absolutely within those elements. This indents the titles as shown in the browser display and right aligns the page numbers.

A table of contents that uses absolute positioning

The HTML for the positioned elements

```
<section>
    <h1><i>Murach&rsquo;s JavaScript and DOM Scripting</i></h1>
    <h2>Section 1<span class="title">Introduction to JavaScript
        programming</span></h2>
    <h3>Chapter 1<span class="title">Introduction to web development and
        JavaScript</span><span class="number">3</span></h3>
    <h3>Chapter 2<span class="title">How to code a JavaScript
        application</span><span class="number">43</span></h3>
    ...
    <h2>Section 2<span class="title">JavaScript essentials</span></h2>
    <h3>Chapter 6<span class="title">How to get input and display
        output</span><span class="number">223</span></h3>
    ...
</section>
```

The CSS for the positioned elements

```
section h2 {
    margin: .6em 0 0;
    position: relative; }
section h3 {
    font-weight: normal;
    margin: .3em 0 0;
    position: relative; }
.title {
    position: absolute;
    left: 90px; }
.number {
    position: absolute;
    right: 0; }
```

Description

- To implement absolute positioning, span elements with class names are used to identify the text to be positioned at the left and the page numbers to be positioned at the right.
- The h2 and h3 elements use relative positioning, but no positions are specified. That way, the .title and .number elements that they contain can be positioned relative to the headings.

Figure 6-14 A table of contents that uses positioning

Perspective

Now that you've completed this chapter, you should be able to develop web pages that have headers, footers, and 2- or 3-column layouts. Better yet, you'll be using CSS to do these layouts, which makes these pages easier to create and maintain. In contrast, if you look at the source code for many web sites today, you'll find that they're still using HTML tables to implement page layouts.

This completes the first section of this book, which we think of as a crash course in HTML and CSS. With these skills, you should be able to develop web pages at a professional level. Then, to add to these skills, you can read any of the chapters in the rest of this book in whatever sequence you prefer. In other words, you can learn those skills whenever you need them.

Terms

float
fixed layout
liquid layout
absolute positioning
relative positioning
fixed positioning

Summary

- When you use the float property to *float* an element, any elements after the floated element will flow into the space left vacant. To make this work, the floated element has to have a width that's either specified or implied.
- To stop an element from flowing into the space left vacant by a floated element, you can use the clear property.
- In a *fixed layout*, the widths of the columns are set. In a *liquid layout*, the width of the page and the width of at least one column change as the user changes the width of the browser window.
- When you use *absolute positioning* for an element, the remaining elements on the page are positioned as if the element weren't there. Because of that, you may need to make room for positioned elements by adjusting other elements.
- When you use *relative positioning* for an element, the remaining elements leave space for the moved element as if it were still there.
- When you use *fixed positioning* for an element, the element doesn't move in the browser window, even when you scroll.

Exercise 6-1 Enhance the Town Hall home page

In this exercise, you'll enhance the formatting of the Town Hall home page that you formatted in exercise 5-1. When you're through, the page should look like this, but without the Speaker of the Month content:

Open the HTML and CSS files

1. Use your text editor to open the HTML and CSS files that you created for exercise 5-1 or 5-2:

```
c:\html5_css3\exercises\town_hall_1\index.html
c:\html5_css3\exercises\town_hall_1\styles\main.css
```

Enhance the HTML and CSS to provide for the two columns

2. In the HTML file, add aside tags so the "Our 2011-2012 Speakers" heading and the headings and images that follow it are within the aside element, and not within the section element. Also, delete the word "Our" from the "Our 2011-2012 Speakers" heading.
3. Still in the HTML file, add the heading and image for the fourth speaker.
4. In the CSS file, enhance the rule set for the body so the width is 800 pixels. Next, set the width of the section to 525 pixels and float it to the right, and set the width of the aside to 215 pixels and float it to the right. Then, use the clear property in the footer to clear the floating. Last, delete the rule set for the Speakers heading. Now, test this. The columns should be starting to take shape.
5. To make this look better, delete the rule set for the Speakers heading. Then, set the left padding for the aside to 20 pixels, and change the right and left padding for the section to 20 pixels. Now, test again.

Get the headings right

6. Add rule sets for the h1 and h2 elements in the aside. They should have the same rules as the h1 and h2 elements in the section.
7. Add a rule set for the img elements in the aside so the bottom padding is set to 1em. Then, test this change.
8. At this point, the page should look good, but it won't include the Speaker of the Month content. Now, make any adjustments, test them in both Firefox and IE, and then go on to the next exercise.

Exercise 6-2 Add the Speaker of the Month

In this exercise, you'll add the Speaker of the Month to the home page. Whenever appropriate, test the changes in Firefox.

Enhance the HTML page

1. Copy the content for the speaker of the month from the file named c6_content.txt in the text folder into the HTML file right before the heading for "Our Ticket Packages".
2. Enclose the Speaker of the Month heading in h1 tags, and enclose the rest of the content in an article element.
3. Within the article, enclose the first heading in h1 tags; enclose the second heading (the date and speaker's name) in h2 tags with a
 tag to provide the line break; and enclose the rest of the text in <p> tags.
4. Add an <a> element within the last paragraph that goes to the sampson.html page in the speakers folder when the user clicks on "Read more." Also, add a non-breaking space and <b> tags so the rest of the line looks right.
5. Add an image element between the h1 and h2 elements in the article, and display the image named sampson_dinosaur.jpg from the images folder.

Enhance the CSS for the home page

6. Add 2 pixel top and bottom borders to the article with #800000 as the color.
7. Float the image in the article to the right, and set its top, bottom, and left margins so there's adequate space around it. Then, add a 1 pixel, black border to the image so the white in the image doesn't fade into the background. Also, delete the rule set for images in the section to get rid of the bottom padding.
8. Change the font color of the h1 elements in the article to to black.
9. Make any final adjustments to the margins or padding, validate the HTML page, and test in IE.

Exercise 6-3 Add one speaker page

In this exercise, you'll add the page for one speaker. This page will be like the home page, but the speaker information will be in the second column as shown below. As you develop this page, test it in Firefox whenever appropriate.

Create the CSS and HTML files for the speaker

1. Copy the index.html file that you've been working with into the speakers folder and name it sampson.html. (With Aptana, you can use the Save As command to save the index file with the new name.)
2. Copy the main.css file that you've been working with into the styles folder and name it speaker.css. (With Aptana, you can the Save As command to do this.)
3. In the head section of the sampson.html file, change the link element so it refers to the speaker.css file in the styles folder. To do that, you can code a document-relative path like this:

```
../styles/speaker.css
```

Modify the HTML file

4. Update all the image and link references to document-relative paths. Then, at the bottom of the HTML aside element, add a link back to the home page within an h2 element.
5. Delete all of the content from the section in the sampson.html file, but not the section tags. Then, copy the text for the speaker from the c6.sampson.txt file in the text folder into the section of the sampson.html file.
6. Enclose the first heading ("Fossil Threads in the Web of Life") within an h1 element. Then, code an image element after the first heading that displays the sampson_dinosaur.jpg file that's in the images folder.
7. Enclose the rest of the content for the speaker in an article element. Within the article, enclose the first heading (the date and speaker's name) in h2 tags with a
 tag to provide the line break, and enclose the rest of the text in <p> tags.
8. Validate the file, and fix any errors.

Modify the CSS file

9. If you test the page right now, it should look pretty good. Then, you just need to adjust the styles to get the page to look the way you want it. For instance, you should delete the borders from the rule set for the article, and add some spacing above the footer.
10. Test all the links to make sure they work correctly. Also, make sure the favicon and all images are displayed correctly.
11. Test the page in IE. Then, if necessary make any corrections, and test again in both Firefox and IE. Now, if you've done everything: "Congratulations!"

Want to review our solution files for this web site?

12. If you had any problems along the way and would like to review our solution files, you'll find them in the folder that follows. The only difference is that the home page already has a navigation bar. This is the web site that you'll be using for the exercises in section 2 of this book:

```
c:\html5_css3\exercises\town_hall_2
```