CHAPTER 2

Structuring HTML and Using Cascading Style Sheets

What You'll Learn in This Chapter:

- How to create a simple web page in HTML
- How to include the HTML tags that every web page must have
- How to use links within your web pages
- How to organize a page with paragraphs and line breaks
- How to organize your content with headings
- How to use the semantic elements of HTML5
- How to begin using basic CSS

In the first chapter, you got a basic idea of the process behind creating web content and viewing it online (or locally, if you do not yet have a web hosting provider). In this chapter, we get down to the business of explaining the various elements that must appear in an HTML file so that it is displayed appropriately in your web browser.

In general, this chapter provides an overview of HTML basics and gives some practical tips to help you make the most of your time as a web developer. You'll begin to dive a bit deeper into the theory behind it all as you learn about the HTML5 elements that enable you to enhance the

semantics—the meaning—of the information that you provide in your marked-up text. You'll take a closer look at six elements that are fundamental to solid semantic structuring of your documents: `<header>`, `<section>`, `<article>`, `<nav>`, `<aside>`, and `<footer>`. Finally, you'll learn the basics of fine-tuning the display of your web content using *Cascading Style Sheets (CSS)*, which enable you to set a number of formatting characteristics, including exact typeface controls, letter and line spacing, and margins and page borders, just to name a few.

Throughout the remainder of this book, you will see HTML tags and CSS styles used appropriately in the code samples, so this chapter makes sure that you have a good grasp of their meaning before we continue.

Getting Started with a Simple Web Page

In the first chapter, you learned that a web page is just a text file that is marked up by (or surrounded by) HTML code that provides guidelines to a browser for displaying the content. To create these text files, use a plain-text editor such as Notepad on Windows or TextEdit on a Mac—do not use WordPad, Microsoft Word, or other full-featured word-processing software because those create different sorts of files than the plain-text files used for web content.

CAUTION

I'll reiterate this point because it is very important to both the outcome and the learning process itself: Do not create your first HTML file with Microsoft Word or any other word processor, even if you can save your file as HTML, because most of these programs will write your HTML for you in strange ways, potentially leaving you totally confused.

Additionally, I recommend that you *not* use a graphical, what-you-see-is-what-you-get (WYSIWYG) editor such as Adobe Dreamweaver. You'll likely find it easier and more educational to start with a simple text editor that forces you to type the code yourself as you're learning HTML and CSS.

Before you begin working, you should start with some text that you want to put on a web page:

1. Find (or write) a few paragraphs of text about yourself, your family, your company, your pets, or some other subject that holds your interest.
2. Save this text as plain, standard ASCII text. Notepad (on Windows) and most simple text editors always save files as plain text, but if you're using another program, you might need to choose this file type as an option (after selecting File, Save As).

As you go through this chapter, you will add HTML markup (called *tags*) to the text file, thus turning it into content that is best viewed in a web browser.

When you save files containing HTML tags, always give them a name ending in `.html`. This is important—if you forget to type the `.html` at the end of the filename when you save the file, most text editors will give it some other extension (such as `.txt`). If that happens, you might not be able to find the file when you try to look at it with a web browser; if you find it, it certainly won't display properly. In other words, web browsers expect a web page file to have a file extension of `.html` and to be in plain-text format.

When visiting websites, you might also encounter pages with a file extension of `.htm`, which is another acceptable file extension to use. You might find other file extensions used on the Web, such as `.jsp` (Java Server Pages), `.aspx` (Microsoft Active Server Pages), and `.php` (PHP: Hypertext Preprocessor). These files also contain HTML in addition to the programming language—although the programming code in those files is executed on the server side and all you would see on the client side is the HTML output. If you looked at the source files, you would likely see some intricate weaving of programming and markup codes. You'll learn more about this process in later chapters as you learn to integrate PHP into your websites.

Listing 2.1 shows an example of text you can type and save to create a simple HTML page. If you opened this file with your web browser, you would see the page shown in Figure 2.1. Every web page you create must include a `<!DOCTYPE>` declaration, as well as `<html></html>`, `<head></head>`, `<title></title>`, and `<body></body>` tag pairs.

LISTING 2.1 The `<html>`, `<head>`, `<title>`, and `<body>` Tags

Click here to view code image

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>The First Web Page</title>
  </head>

  <body>
    <p>
      In the beginning, Tim created the HyperText Markup Language.
The
      Internet was without form and void, and text was upon the
face of
      the monitor and the Hands of Tim were moving over the face of
the
      keyboard. And Tim said, Let there be links; and there were
links.
      And Tim saw that the links were good; and Tim separated the
links
      from the text. Tim called the links Anchors, and the text He
      called Other Stuff. And the whole thing together was the
first
      Web Page.
    </p>
  </body>
</html>
```

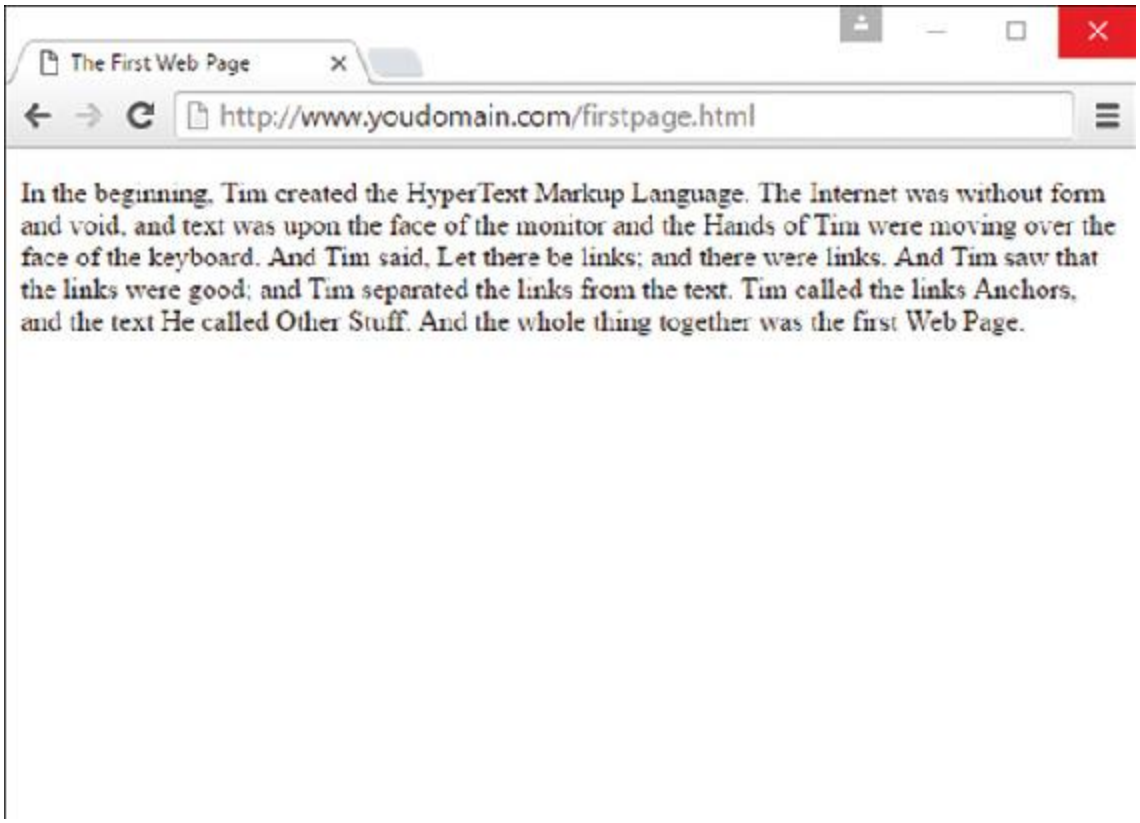

FIGURE 2.1
When you save the text in Listing 2.1 as an HTML file and view it with a web browser, only the actual title and body text are displayed.

In Listing 2.1, as in every HTML page, the words starting with < and ending with > are actually coded commands. These coded commands are called *HTML tags* because they "tag" pieces of text and tell the web browser what kind of text it is. This allows the web browser to display the text appropriately.

The first line in the document is the document type declaration; you are *declaring* that it is HTML (specifically, HTML5) because `html` is the value used to declare a document as HTML5 in the `<!DOCTYPE>` tag.

▼ TRY IT YOURSELF

Creating and Viewing a Basic Web Page

Before you learn the meaning of the HTML tags used in Listing 2.1, you might want to see exactly how I went about creating and viewing the document itself. Follow these steps:

1. Type all the text in Listing 2.1, including the HTML tags, in Windows Notepad (or use Macintosh TextEdit or another text editor of your choice).
2. Select File, Save As. Be sure to select plain text (or ASCII text) as the file type.
3. Name the file **`firstpage.html`**.
4. Choose the folder on your hard drive where you want to keep your web pages—and remember which folder you choose! Click the Save or OK button to save the file.
5. Now start your favorite web browser. (Leave Notepad running, too, so you can easily switch between viewing and editing your page.)

In Internet Explorer, select File, Open and click Browse. If you're using Firefox, select File, Open File. Navigate to the appropriate folder and select the `firstpage.html` file. Some browsers and operating systems also enable you to drag and drop the `firstpage.html` file onto the browser window to view it.

Voila! You should see the page shown in Figure 2.1.

If you have obtained a web hosting account, you could use FTP at this point to transfer the `firstpage.html` file to the web server. In fact, from this chapter forward, the instructions assume that you have a hosting provider and are comfortable sending files back and forth via FTP; if that is not the case, you should review the first chapter before moving on. Alternatively, if you are consciously choosing to work with files locally (without a web host), be prepared to adjust the instructions to suit your particular needs (such as ignoring the commands "transfer the files" and "type in the URL").

NOTE

You don’t need to be connected to the Internet to view a web page stored on your own computer. By default, your web browser probably tries to connect to the Internet every time you start it, which makes sense most of the time. However, this can be a hassle if you’re developing pages locally on your hard drive (offline) and you keep getting errors about a page not being found. If you have a full-time web connection via a cable modem, DSL, or Wi-Fi, this is a moot point because the browser will never complain about being offline. Otherwise, the appropriate action depends on your breed of browser; check the options under your browser’s Tools menu.

HTML Tags Every Web Page Must Have

The time has come for the secret language of HTML tags to be revealed to you. When you understand this language, you will have creative powers far beyond those of other humans. Don’t tell the other humans, but it’s really pretty easy.

The first line of code is the document type declaration; in HTML5, this is simply `<!DOCTYPE html>`.

This declaration identifies the document as being HTML5, which then ensures that web browsers know what to expect and can prepare to render content in HTML5.

Many HTML tags have two parts: an *opening tag*, which indicates where a piece of text begins, and a *closing tag*, which indicates where the piece of text ends. Closing tags start with a forward slash (/) just after the < symbol.

Another type of tag, or element, is the *empty element*, which is different in that it doesn’t include a pair of matching opening and closing tags. Instead, an empty element consists of a single tag that starts with the < symbol and ends with the > symbol. You may see some empty elements end with />, which is no longer required in HTML5 but did exist in previous versions of HTML.

Following is a quick summary of these three types of tags, just to make sure you understand the role each plays:

- An *opening tag* is an HTML tag that indicates the start of an HTML command; the text affected by the command appears after the opening tag. Opening tags always begin with < and end with >, as in `<html>`.
- A *closing tag* is an HTML tag that indicates the end of an HTML command; the text affected by the command appears before the closing tag. Closing tags always begin with `</` and end with >, as in `</html>`.
- An *empty tag* or *empty element* is an HTML tag that issues an HTML command without enclosing any text in the page. Examples include `<br>` for line breaks and `<img>` for images.

NOTE

You no doubt noticed in Listing 2.1 that there is some extra code associated with the `<html>` tag. This code consists of the language attribute (`lang`), which is used to specify additional information related to the tag. In this case, it specifies that the language of the text within the HTML is English. If you are writing in a different language, replace the `en` (for English) with the language identifier relevant to you.

For example, the `<body>` tag in Listing 2.1 tells the web browser where the actual body text of the page begins, and `</body>` indicates where it ends. Everything between the `<body>` and `</body>` tags appears in the main display area of the web browser window, as shown in Figure 2.1.

The very top of the browser window (refer to Figure 2.1) shows title text, which is any text that is located between `<title>` and `</title>`. The title text also identifies the page on the browser's Bookmarks or Favorites menu, depending on which browser you use. It's important to provide titles for your pages so that visitors to the page can properly bookmark them for future reference; search engines also use titles to provide a link to search results.

You will use the <body> and <title> tag pairs in every HTML page you create because every web page needs a title and body text. You will also use the <html> and <head> tag pairs, which are the other two tags shown in Listing 2.1. Putting <html> at the very beginning of a document simply indicates that the document is a web page. The </html> at the end indicates that the web page is over.

Within a page, there is a head section and a body section. Each section is identified by <head> and <body> tags. The idea is that information in the head of the page somehow describes the page but isn't actually displayed by a web browser. Information placed in the body, however, is displayed by a web browser. The <head> tag always appears near the beginning of the HTML code for a page, just after the opening <html> tag.

TIP

You might find it convenient to create and save a bare-bones page (also known as a *skeleton* page, or *template*) with just the DOCTYPE and opening and closing <html>, <head>, <title>, and <body> tags, similar to the document in Listing 2.1. You can then open that document as a starting point whenever you want to make a new web page and save yourself the trouble of typing all those obligatory tags every time.

The <title> tag pair used to identify the title of a page appears within the head of the page, which means it is placed after the opening <head> tag and before the closing </head> tag. In upcoming lessons, you'll learn about some other advanced header information that can go between <head> and </head>, such as style sheet rules for formatting the page.

The tag pair in Listing 2.1 encloses a paragraph of text. You should enclose your chunks of text in the appropriate container elements whenever possible; you'll learn more about container elements as you move forward in your lessons.

Using Hyperlinks in Web Pages

There is no rule that says you have to include links in your web content, but you would be hard-pressed to find a website that doesn't include at least one link either to another page on the same domain (for example, yourdomain.com), another domain, or even the same page. Links are all over the web, but it is important to understand a little bit of the "under the hood" details of links.

When files are part of the same domain, you can link to them by simply providing the name of the file in the `href` attribute of the `<a>` tag. An *attribute* is an extra piece of information associated with a tag that provides further details about the tag. For example, the `href` attribute of the `<a>` tag identifies the address of the page to which you are linking.

When you have more than a few pages, or when you start to have an organizational structure to the content in your site, you should put your files into directories (or *folders*, if you will) whose names reflect the content within them. For example, all your images could be in an `images` directory, company information could be in an `about` directory, and so on. Regardless of how you organize your documents within your own web server, you can use relative addresses, which include only enough information to find one page from another. A *relative address* describes the path from one web page to another, instead of a full (or *absolute*) Internet address which includes the full protocol (`http` or `https`) and the domain name (like www.yourdomain.com).

As you recall from Chapter 1, "Understanding How the Web Works," the document root of your web server is the directory designated as the top-level directory for your web content. In web addresses, that document root is represented by the forward slash (`/`). All subsequent levels of directories are separated by the same type of forward slash. Here's an example:

Click here to view code image

```
/directory/subdirectory/subsubdirectory/
```

CAUTION

The forward slash (/) is always used to separate directories in HTML. Don't use the backslash (\, which is normally used in Windows) to separate your directories. Remember, everything on the Web moves forward, so use forward slashes.

Suppose you are creating a page named `zoo.html` in your document root, and you want to include a link to pages named `african.html` and `asian.html` in the `elephants` subdirectory. The links would look like the following:

Click here to view code image

```
<a href="/elephants/african.html">Learn about African elephants.
</a>
<a href="/elephants/asian.html">Learn about Asian elephants.</a>
```

Linking Within a Page Using Anchors

The <a> tag—the tag responsible for hyperlinks on the Web—got its name from the word *anchor*, because a link serves as a designation for a spot in a web page. The <a> tag can be used to mark a spot on a page as an anchor, enabling you to create a link that points to that exact spot. For example, the top of a page could be marked as:

```
<a name="top"></a>
```

The <a> tag normally uses the `href` attribute to specify a hyperlinked target. The <a href> is what you click, and <a id> is where you go when you click there. In this example, the <a> tag is still specifying a target, but no actual link is created that you can see. Instead, the <a> tag gives a name to the specific point on the page where the tag occurs. The tag must be included and a unique name must be assigned to the `id` attribute, but no text between <a> and is necessary.

To link to this location, you would use the following:

Click here to view code image

```
<a href="#top">Go to Top of Page</a>
```

Linking to External Web Content

The only difference between linking to pages within your own site and linking to external web content is that when linking outside your site, you need to include the full address to that content. The full address includes the `http://` before the domain name and then the full pathname to the file (for example, an HTML file, an image file, or a multimedia file).

For example, to include a link to Google from within one of your own web pages, you would use this type of absolute addressing in your `<a>` link:

Click here to view code image

```
<a href="http://www.google.com/">Go to Google</a>
```

CAUTION

As you might know, you can leave out the `http://` at the front of any address when typing it into most web browsers. However, you *cannot* leave that part out when you type an Internet address into an `<a href>` link on a web page.

You can apply what you learned in previous sections to creating links to named anchors on other pages. Linked anchors are not limited to the same page. You can link to a named anchor on another page by including the address or filename followed by `#` and the anchor name. For example, the following link would take you to an anchor named `photos` within the african.html page inside the `elephants` directory on the (fictional) domain www.takeme2thezoo.com:

Click here to view code image

```
<a
href="http://www.takeme2thezoo.com/elephants/african.html#photos">
Check out the African Elephant Photos!</a>
```

If you are linking from another page already on the www.takeme2thezoo.com domain (because you are, in fact, the site maintainer), your link might simply be as follows:

Click here to view code image

```
<a href="/elephants/african.html#photos">Check out the
African Elephant Photos!</a>
```

The `http://` and the domain name would not be necessary in that instance, as you have already learned.

CAUTION

Be sure to include the `#` symbol only in `<a href>` link tags. Don't put the `#` symbol in the `<a id>` tag; links to that name won't work in that case.

Linking to an Email Address

In addition to linking between pages and between parts of a single page, the `<a>` tag enables you to link to email addresses. This is the simplest way to enable your web page visitors to talk back to you. Of course, you could just provide visitors with your email address and trust them to type it into whatever email programs they use, but that increases the likelihood for errors. By providing a clickable link to your email address, you make it almost completely effortless for them to send you messages and eliminate the chance for typos.

An HTML link to an email address looks like the following:

Click here to view code image

```
<a href="mailto:yourusername@yourdomain.com">Send me an
email message.</a>
```

The words `Send me an email message` will appear just like any other `<a>` link.

Having taken this brief foray into the world of hyperlinks, let's get back to content organization and display.

Organizing a Page with Paragraphs and Line Breaks

When a web browser displays HTML pages, it pays no attention to line endings or the number of spaces between words in the underlying text file itself. For example, the top version of the poem in Figure 2.2 appears with a single space between all words, even though that's not how it's shown in Listing 2.2. This is because extra whitespace in HTML code is automatically reduced to a single space when rendered by the web browser. Additionally, when the text reaches the edge of the browser window, it automatically wraps to the next line, no matter where the line breaks were in the original HTML file.

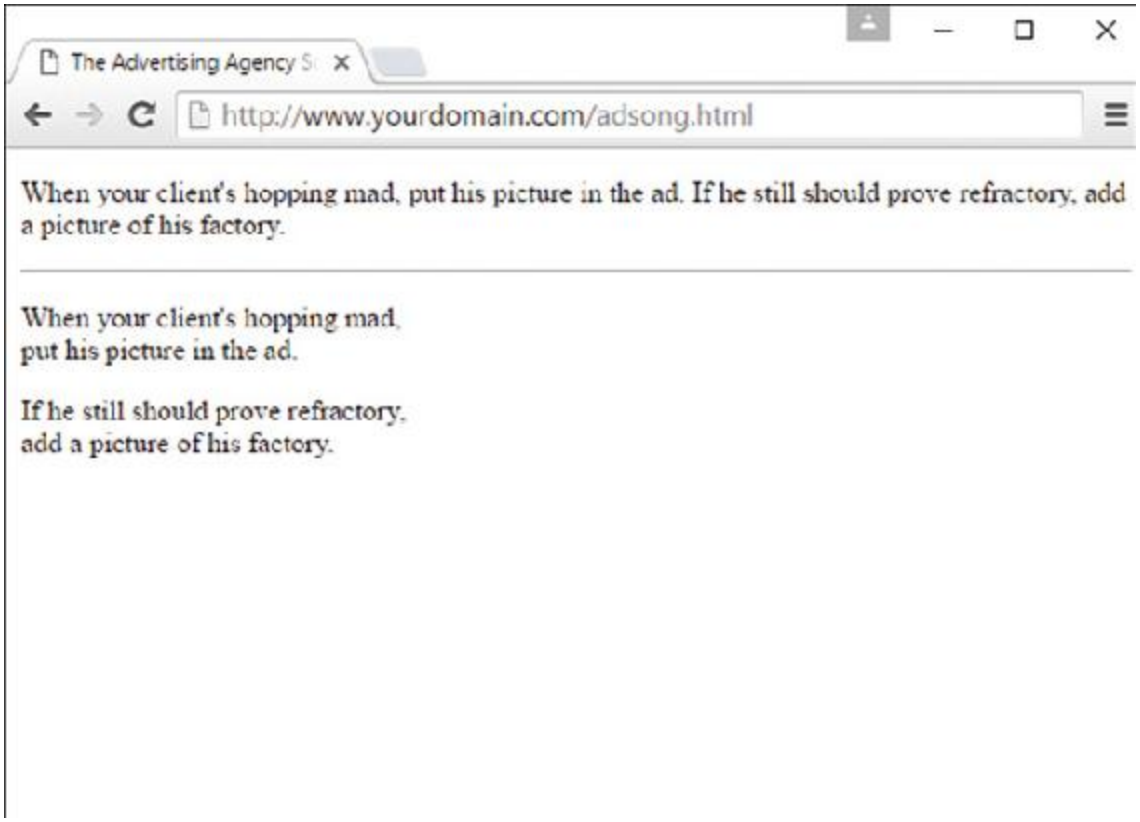

FIGURE 2.2
When the HTML in Listing 2.2 is viewed as a web page, line and paragraph breaks appear only where there are `<br>` and `<p>` tags.

LISTING 2.2 HTML Containing Paragraph and Line Breaks

Click here to view code image

```
<!DOCTYPE html>

<html lang="en">
  <head>
    <title>The Advertising Agency Song</title>
  </head>

  <body>
    <p>
      When your client's    hopping mad,
      put his picture in the ad.

      If he still should    prove refractory,
      add a picture of his factory.
    </p>
    <hr>
    <p>
      When your client's hopping mad,<br>
      put his picture in the ad.
    </p>
    <p>
      If he still should prove refractory,<br>
      add a picture of his factory.
    </p>
  </body>
</html>
```

You must use HTML tags if you want to control where line and paragraph breaks actually appear. When text is enclosed within the container tags, a line break is assumed after the closing tag. In later chapters, you'll learn to control the height of the line break using CSS. The
 tag forces a line break within a paragraph. Unlike the other tags you've seen so far,
 doesn't require a closing </br> tag—this is one of those empty elements discussed earlier.

The poem in Listing 2.2 and Figure 2.2 shows the
 and tags used to separate the lines and verses of an advertising agency song. You might have also noticed the <hr> tag in the listing, which causes a horizontal rule line to appear on the page (see Figure 2.2). Inserting a horizontal rule with the <hr> tag also causes a line break, even if you don't include a
 tag along with it. Like
, the <hr> horizontal rule tag is an empty element and, therefore, never gets a closing </hr> tag.

▼ TRY IT YOURSELF

Formatting Text in HTML

Try your hand at formatting a passage of text as proper HTML:

1. Add `<html><head><title>` *`My Title`* `</title></head><body>` to the beginning of the text (using your own title for your page instead of *My Title*). Also include the boilerplate code at the top of the page that takes care of meeting the requirements of standard HTML.
2. Add `</body></html>` to the very end of the text.
3. Add a `<p>` tag at the beginning of each paragraph and a `</p>` tag at the end of each paragraph.
4. Use `<br>` tags anywhere you want single-spaced line breaks.
5. Use `<hr>` to draw horizontal rules separating major sections of text, or wherever you'd like to see a line across the page.
6. Save the file as *`mypage`*`.html` (using your own filename instead of *`mypage`*).

CAUTION

If you are using a word processor to create the web page, be sure to save the HTML file in plain text or ASCII format.

7. Open the file in a web browser to see your web content. (Send the file via FTP to your web hosting account, if you have one.)
8. If something doesn't look right, go back to the text editor to make corrections and save the file again (and send it to your web hosting account, if applicable). You then need to click Reload/Refresh in the browser to see the changes you made.

Organizing Your Content with Headings

When you browse web pages on the Internet, you'll notice that many of them have a heading at the top that appears larger and bolder than the rest of the text. Listing 2.3 is sample code and text for a simple web page containing an example of a heading as compared to normal paragraph text. Any text between the `<h1>` and `</h1>` tags will appear as a large heading. Additionally, `<h2>` and `<h3>` make progressively smaller headings, all the way down to `<h6>`.

LISTING 2.3 Using Heading Tags

Click here to view code image

```
<!DOCTYPE html>

<html lang="en">
  <head>
    <title>My Widgets</title>
  </head>

  <body>
    <h1>My Widgets</h1>
    <p>My widgets are the best in the land. Continue reading to
    learn more about my widgets.</p>

    <h2>Widget Features</h2>
    <p>If I had any features to discuss, you can bet I'd do
    it here.</p>

    <h3>Pricing</h3>
    <p>Here, I would talk about my widget pricing.</p>

    <h3>Comparisons</h3>
    <p>Here, I would talk about how my widgets compare to my
    competitor's widgets.</p>
  </body>
</html>
```

NOTE

By now, you've probably caught on to the fact that HTML code is often indented by its author to reveal the relationship between different parts of

the HTML document, as well as for simple ease of reading. This indentation is entirely voluntary—you could just as easily run all the tags together with no spaces or line breaks, and they would still look fine when viewed in a browser. The indentations are for you so that you can quickly look at a page full of code and understand how it fits together. Indenting your code is another good web design habit and ultimately makes your pages easier to maintain, both for yourself and for anyone else who might pick up where you leave off.

As you can see in Figure 2.3, the HTML that creates headings couldn't be simpler. In this example, the phrase "My Widgets" is given the highest level of heading, and is prominently displayed, by surrounding it with the `<h1>` `</h1>` tag pair. For a slightly smaller (level 2) heading—for information that is of lesser importance than the title—use the `<h2>` and `</h2>` tags around your text. For content that should appear even less prominently than a level 2 heading, use the `<h3>` and `</h3>` tags around your text.

However, bear in mind that your headings should follow a content hierarchy; use only one level 1 heading, have one (or more) level 2 headings after the level 1 heading, use level 3 headings only after level 2 headings, and so on. Do not fall into the trap of assigning headings to content just to make that content display a certain way, such as by skipping headings. Instead, ensure that you are categorizing your content appropriately (as a main heading, a secondary heading, and so on) while using display styles to make that text render a particular way in a web browser.

You can also use `<h4>`, `<h5>`, and `<h6>` tags to make progressively less important headings. By default, web browsers seldom show a noticeable difference between these headings and the `<h3>` headings—although you can control that with your own CSS. Also, content usually isn't displayed in such a manner that you'd need six levels of headings to show the content hierarchy.

It's important to remember the difference between a *title* and a *heading*. These two words are often interchangeable in day-to-day English, but when you're talking HTML, `<title>` gives the entire page an identifying name

that isn't displayed on the page itself; it's displayed only on the browser window's title bar. The heading tags, on the other hand, cause some text on the page to be displayed with visual emphasis. There can be only one `<title>` per page, and it must appear within the `<head>` and `</head>` tags; on the other hand, you can have as many `<h1>`, `<h2>`, and `<h3>` headings as you want, in any order that suits your fancy. However, as I mentioned before, you should use the heading tags to keep tight control over content hierarchy (logic dictates only one `<h1>` heading); do not use headings as a way to achieve a particular look, because that's what CSS is for.

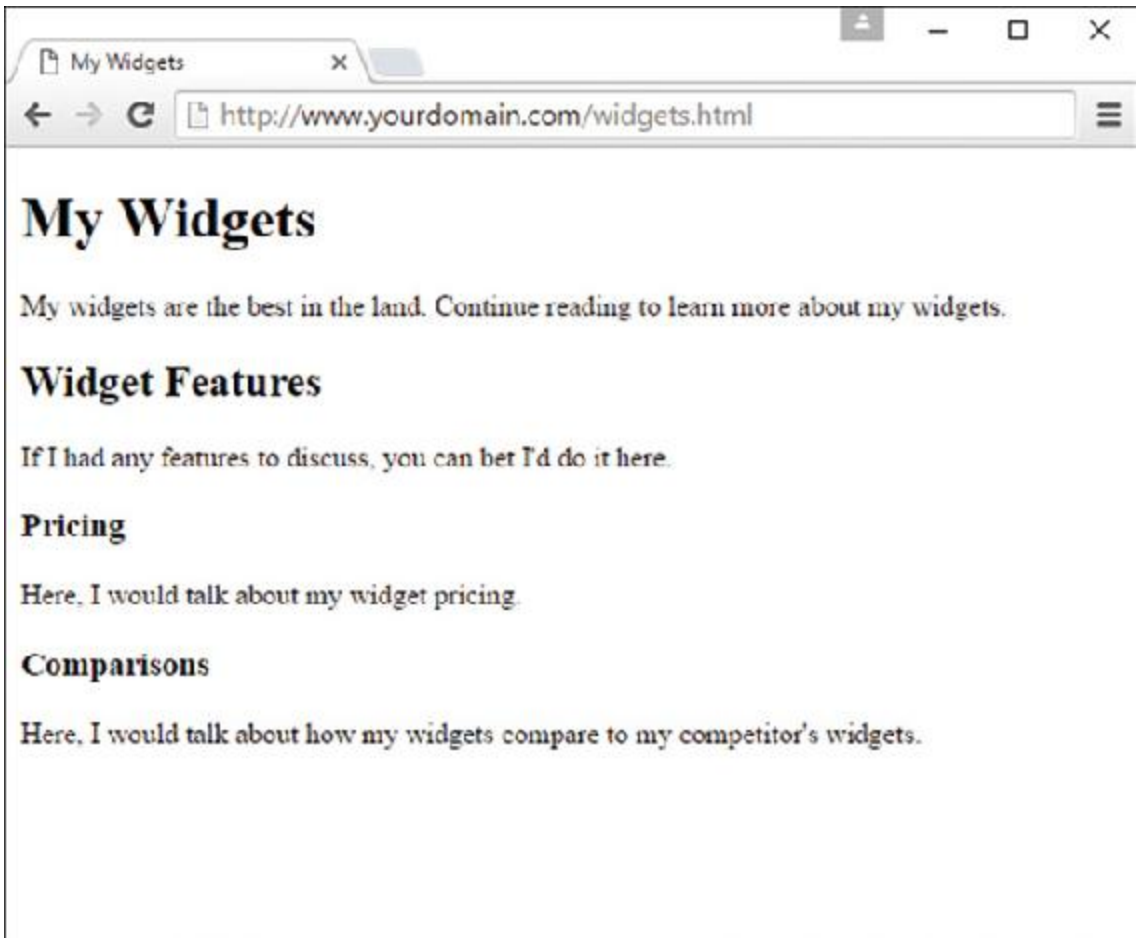

FIGURE 2.3
Using three levels of headings shows the hierarchy of content on this sample product page.

CAUTION

Don’t forget that anything placed in the head of a web page is not intended to be viewed on the page, whereas everything in the body of the page *is* intended for viewing.

Peeking at Other Designers’ Pages

Given the visual and sometimes audio pizzazz present in many popular web pages, you probably realize that the simple pages described in this lesson are only the tip of the HTML iceberg. Now that you know the basics, you might surprise yourself with how much of the rest you can pick up just by looking at other people’s pages on the Internet. You can see the HTML for any page by right-clicking and selecting View Source in any web browser.

Don’t worry if you aren’t yet able to decipher what some HTML tags do or exactly how to use them yourself. You’ll find out about all those things as you move forward in the book. However, sneaking a preview now will show you the tags that you do know in action and give you a taste of what you’ll soon be able to do with your web pages.

Understanding Semantic Elements

HTML5 includes tags that enable you to enhance the semantics—the meaning—of the information you provide in your marked-up text. Instead of simply using HTML as a presentation language, as was the practice in the very early days when `<b>` for bold and `<i>` for italics was the norm, modern HTML has as one of its goals the separation of presentation and meaning. While using CSS to provide guidelines for presentation, composers of HTML can provide meaningful names within their markup for individual elements, not only through the use of IDs and class names (which you’ll learn about later in this chapter), but also through the use of semantic elements.

Some of the semantic elements available in HTML5 follow:

- **<header></header>**—This might seem counterintuitive, but you can use multiple <header> tags within a single page. The <header> tag should be used as a container for introductory information, so it might be used only once in your page (likely at the top), but you also might use it several times if your page content is broken into sections. Any container element can have a <header> element; just make sure that you're using it to include introductory information about the element it is contained within.
- **<footer></footer>**—The <footer> tag is used to contain additional information about its containing element (page or section), such as copyright and author information or links to related resources.
- **<nav></nav>**—If your site has navigational elements, such as links to other sections within a site or even within the page itself, these links go in a <nav> tag. A <nav> tag typically is found in the first instance of a <header> tag, just because people tend to put navigation at the top and consider it introductory information—but that is not a requirement. You can put your <nav> element anywhere (as long as it includes navigation), and you can have as many on a page as you need (often no more than two, but you might feel otherwise).
- **<section></section>**—The <section> tag contains anything that relates thematically; it can also contain a <header> tag for introductory information and possibly a <footer> tag for other related information. You can think of a <section> as carrying more meaning than a standard <p> (paragraph) or (division) tag, which typically conveys no meaning at all; the use of <section> conveys a relationship between the content elements it contains.
- **<article></article>**—An <article> tag is like a <section> tag, in that it can contain a <header>, a <footer>, and other container elements such as paragraphs and divisions. But the additional meaning carried with the <article> tag is that it is, well, like an article in a newspaper or some other publication. Use this tag around blog posts, news articles, reviews, and other items that fit this description. One key difference between an <article> and a <section> is that an <article> is a standalone body of work, whereas a <section> is a thematic grouping of information.

- **<aside></aside>**—Use the <aside> tag to indicate secondary information; if the <aside> tag is within a <section> or an <article>, the relationship will be to those containers; otherwise, the secondary relationship will be to the overall page or site itself. It might make sense to think of the <aside> as a sidebar—either for all the content on the page or for an article or other thematic container of information.

These semantic elements will become clearer as you practice using them. In general, using semantic elements is a good idea because they provide additional meaning not only for you and other designers and programmers reading and working with your markup, but also for machines. Web browsers and screen readers will respond to your semantic elements by using them to determine the structure of your document; screen readers will report a deeper meaning to users, thus increasing the accessibility of your material.

One of the best ways to understand the HTML5 semantic elements is to see them in action, but that can be a little difficult when the primary purpose of these elements is to provide *meaning* rather than design. That's not to say that you can't add design to these elements—you most certainly can. But the "action" of the semantic elements is to hold content and provide meaning through doing so, as in Figure 2.4, which shows a common use of semantic elements for a basic web page.

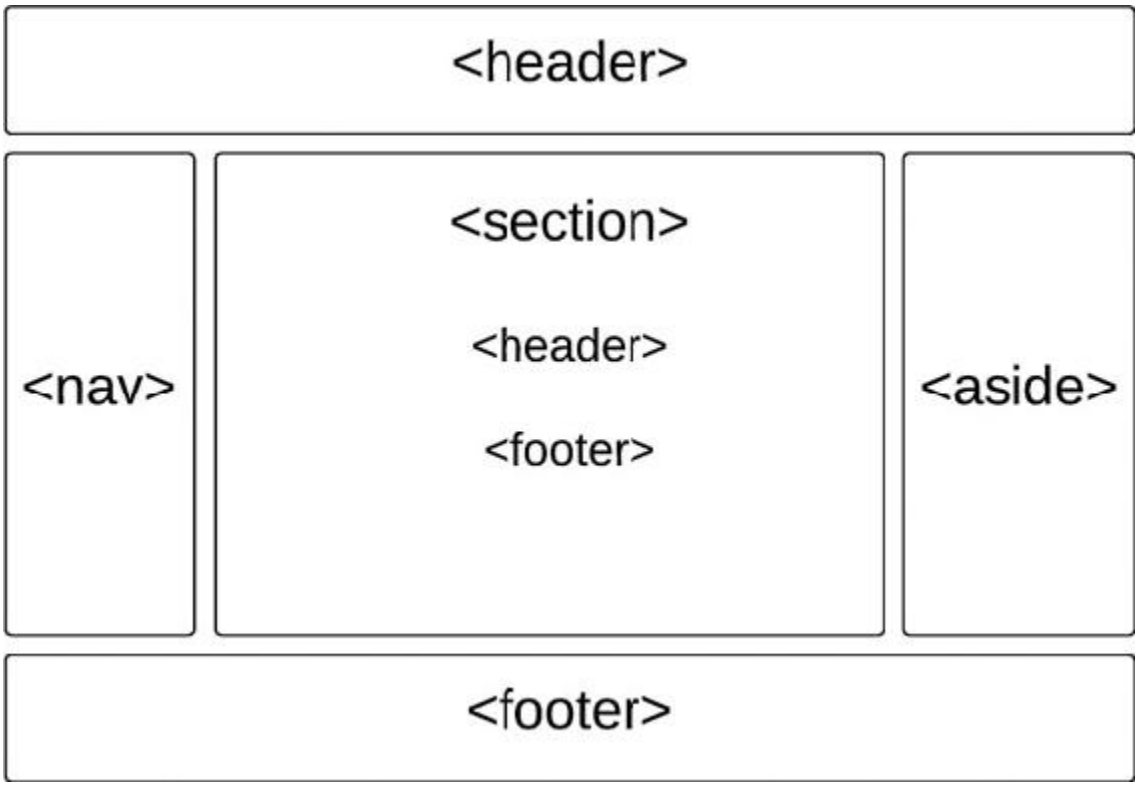

FIGURE 2.4
Showing basic semantic elements in a web page.

Initially, you might think, "Of *course*, that makes total sense, with the header at the top and the footer at the bottom," and feel quite good about yourself for understanding semantic elements at first glance—and you should! A second glance should then raise some questions: What if you want your navigation to be horizontal under your header? Does an aside have to be (literally) on the side? What if you don't want any asides? What's with the use of `<header>` and `<footer>` again within the main body section? And that's just to name a few! Something else you might wonder about is where the `<article>` element fits in; it isn't shown in this example but will be used later in this chapter.

This is the time when conceptualizing the page—and specifically the page *you* want to create—comes into play. If you understand the content you want to mark up and you understand that you can use any, all, or none of the semantic elements and still create a valid HTML document, then you can begin to organize the content of your page in the way that makes the most sense for it and for you (and, hopefully, for your readers).

NOTE

Although you do not need to use semantic elements to create a valid HTML document, even a minimal set is recommended so that web browsers and screen readers can determine the structure of your document. Screen readers are capable of reporting a deeper meaning to users, thus increasing the accessibility of your material.

(If this note were marked up in an HTML document, it would use the `<aside>` element.)

Let's take a look at the elements used in Figure 2.4 before moving on to a second example and then a deeper exploration of the individual elements themselves. In Figure 2.4, you see a `<header>` at the top of the page and a `<footer>` at the bottom—straightforward, as already mentioned. The use of a `<nav>` element on the left side of the page matches a common display area for navigation, and the `<aside>` element on the right side of the page matches a common display area for secondary notes, pull quotes, helper text, and "for more information" links about the content. In Figure 2.5, you'll see some of these elements shifted around, so don't worry—Figure 2.4 is not some immutable example of semantic markup.

Something you might be surprised to see in Figure 2.5 is the `<header>` and `<footer>` inside the `<section>` element. As you'll learn shortly, the role of the `<header>` element is to introduce a second example and then a deeper exploration of the individual elements themselves. In Figure 2.4, you see a `<header>` at the top of the page and a `<footer>` at the bottom—straightforward, as already mentioned. The use of a `<nav>` element on the left side of the page matches the content that comes after it, and the `<header>` element itself does not convey any level in a document outline. Therefore, you can use as many as you need to mark up your content appropriately; a `<header>` at the beginning of the page might contain introductory information about the page as a whole, and the `<header>` element within the `<section>` element might just as easily and appropriately contain introductory information about the content within it. The same is true for the multiple appearances of the `<footer>` element in this example.

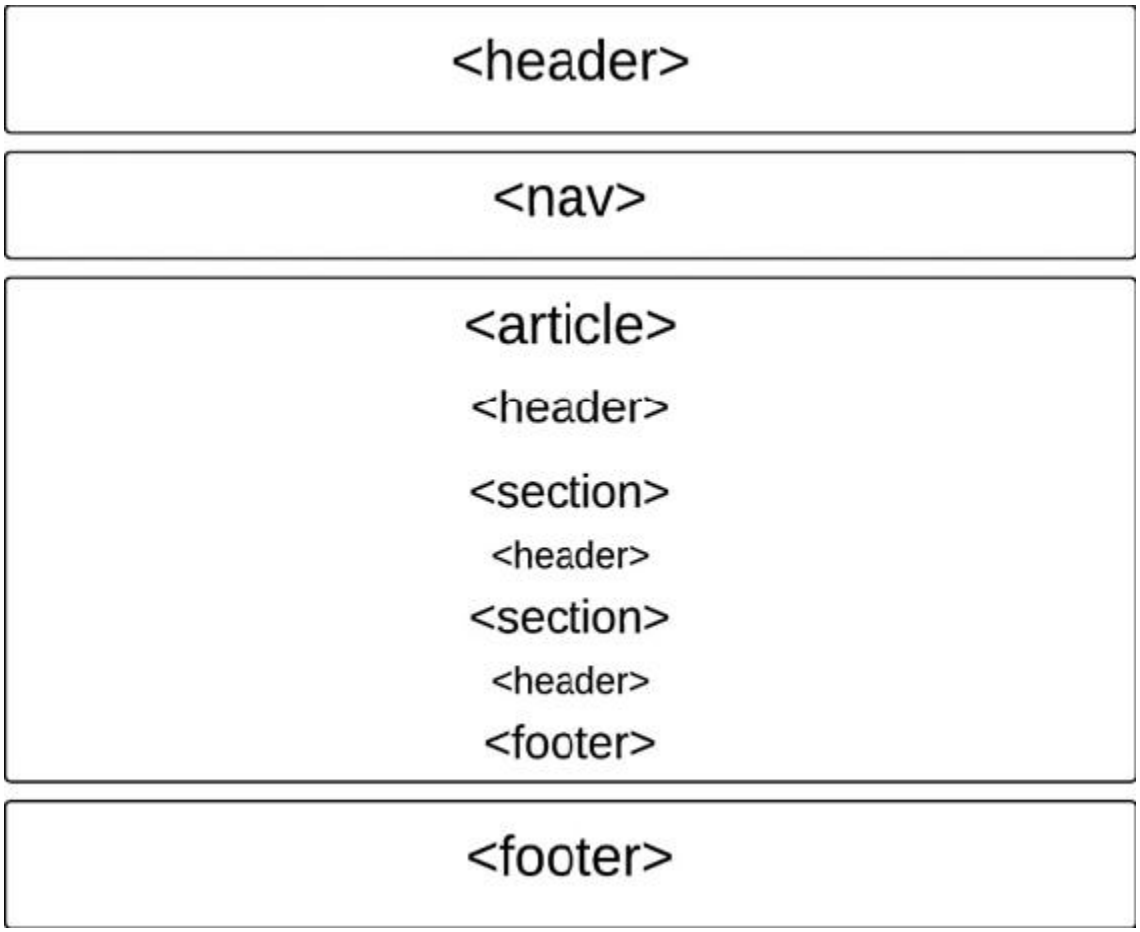

FIGURE 2.5
Using nested semantic elements to add more meaning to the content.

Let's move on to Figure 2.5, which shifts around the `<nav>` element and also introduces the use of the `<article>` element.

In Figure 2.5, the `<header>` and `<nav>` elements at the beginning of the page, and the `<footer>` element at the bottom of the page, should make perfect sense to you. And, although we haven't talked about the `<article>` element yet, if you think about it as a container element that has sections (`<section>`s, even!), with each of those sections having its own heading, then the chunk of semantic elements in the middle of the figure should make sense, too. As you can see, there's no single way to conceptualize a page; you should conceptualize content, and that content will be different on each page in a web site.

If you marked up some content in the structure shown in Figure 2.5, it might look like Listing 2.4.

LISTING 2.4 Semantic Markup of Basic Content

Click here to view code image

```
<!DOCTYPE html>

<html lang="en">
  <head>
    <title>Semantic Example</title>
  </head>
  <body>
    <header>
       <h1>SITE OR PAGE LOGO GOES HERE</h1>
    </header>
    <nav>
       SITE OR PAGE NAV GOES HERE.
    </nav>
    <article>
       <header>
          <h2>Article Heading</h2>
       </header>
       <section>
          <header>
             <h3>Section 1 Heading</h3>
          </header>
          <p>Section 1 content here.</p>
       </section>
       <section>
          <header>
             <h3>Section 2 Heading</h3>
          </header>
          <p>Section 2 content here.</p>
       </section>
       <footer>
          <p>Article footer goes here.</p>
       </footer>
    </article>
    <footer>
       SITE OR PAGE FOOTER HERE
    </footer>
  </body>
</html>
```

If you opened this HTML document in your web browser, you would see something like what's shown in Figure 2.6—a completely unstyled

document, but one that has semantic meaning (even if no one can “see” it).

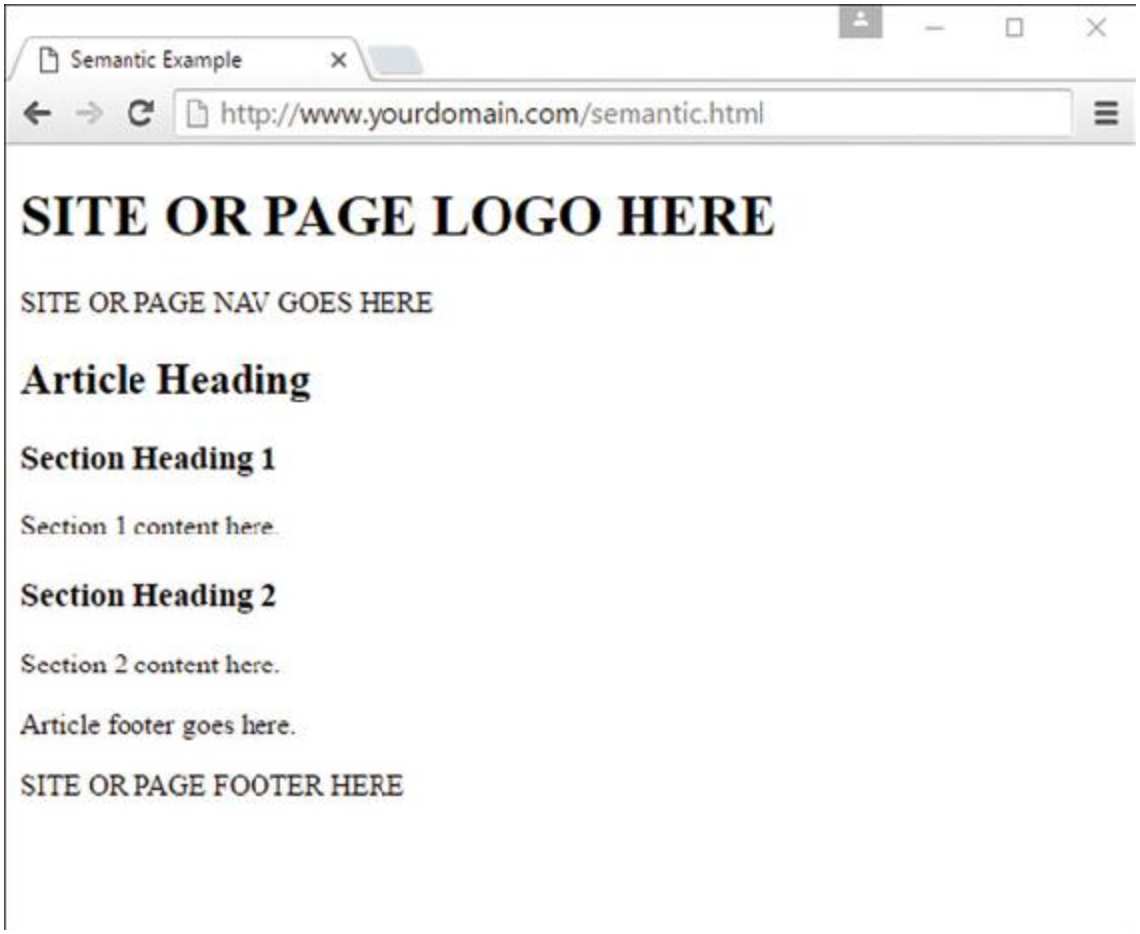

FIGURE 2.6
The output of Listing 2.4.

Just because there is no visible styling doesn’t mean the meaning is lost; as noted earlier in this section, machines can interpret the structure of the document as provided for through the semantic elements. You can see the outline of this basic document in Figure 2.7, which shows the output of this file after examination by the HTML5 Outline tool at http://gsnedders.html5.org/outliner/.

TIP

Using the HTML5 Outline tool is a good way to check that you’ve created your headers, footers, and sections; if you examine your document and see “untitled section” anywhere, and those untitled sections do not match up

with a <nav> or <aside> element (which has more relaxed guidelines about containing headers), then you have some additional work to do.

Now that you've seen some examples of conceptualizing the information represented in your documents, you're better prepared to start marking up those documents. The sections that follow take a look at the semantic elements individually.

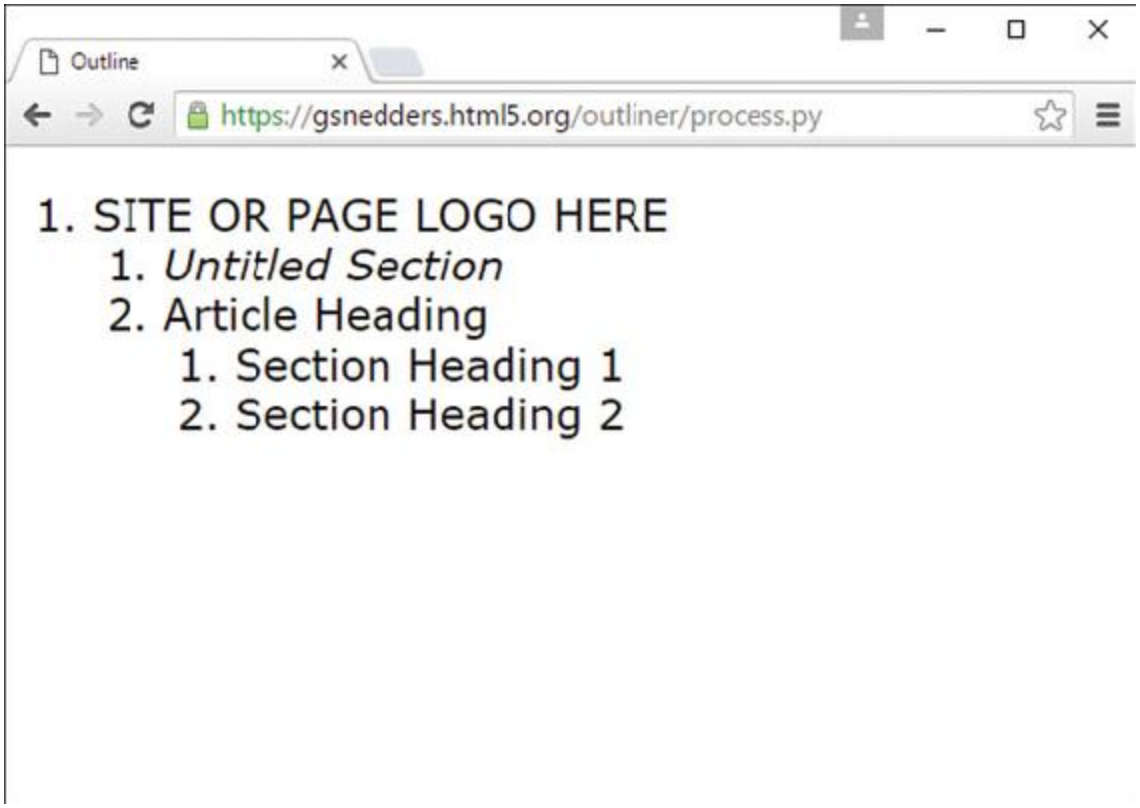

FIGURE 2.7
The outline of this document follows the semantic markup.

Using <header> in Multiple Ways

At the most basic level, the <header> element contains introductory information. That information might take the form of an actual <h1> (or other level) element, or it might simply be a logo image or text contained within a <p> or element. The *meaning* of the content should be introductory in nature, to warrant its inclusion within a <header> </header> tag pair.

As you've seen in the examples so far in this lesson, a common placement of a <header> element is at the beginning of a page. When it's used in this way, containing a logo or an <h1>-level title makes sense, such as here:

Click here to view code image

```
<header>
     <img src="acmewidgets.jpg" alt="ACME Widgets LLC">
</header>
```

Or even here:

Click here to view code image

```
<header>
     <img src="acmewidgets.jpg" alt="ACME Widgets LLC">
     <h1>The finest widgets are made here!</h1>
</header>
```

Both snippets are valid uses of <header> because the information contained within them is introductory to the page overall.

As you've also seen in this chapter, you are not limited to only one <header>. You can go crazy with your <header> elements, as long as they are acting as containers for introductory information—Listing 2.4 showed the use of <header> elements for several <section> elements within an <article>, and this is a perfectly valid use of the element:

Click here to view code image

```
<section>
   <header>
      <h3>Section 1 Heading</h3>
   </header>
   <p>Section 1 content here.</p>
</section>
<section>
   <header>
      <h3>Section 2 Heading</h3>
   </header>
   <p>Section 2 content here.</p>
</section>
```

The <header> element can contain any other element in the flow content category, of which it is also a member. This means that a <header> *could* contain a <section> element, if you wanted, and be perfectly valid markup. However, when you are conceptualizing your content, think about whether that sort of nesting makes sense before you go off and do it.

NOTE

In general, *flow content* elements are elements that contain text, images, or other multimedia embedded content; HTML elements fall into multiple categories.

If you want to learn more about the categorization of elements into content models, see http://www.w3.org/TR/2011/WD-html5-20110525/content-models.html.

The only exceptions to the permitted content within <header> are that the <header> element cannot contain *other* <header> elements and it cannot contain a <footer> element. Similarly, the <header> element cannot be contained within a <footer> element.

Understanding the <section> Element

The <section> element has a simple definition: It is a "generic section of a document" that is also a "thematic grouping of content, typically with a heading." That sounds pretty simple to me, and probably does to you as well. So you might be surprised to find that if you type "difference between section and article in HTML5" in your search engine of choice, you'll find tens of thousands of entries talking about the differences because the definitions trip people up all the time. We first discuss the <section> element and then cover the <article> element—and hopefully avoid any of the misunderstandings that seem to plague new web developers.

In Listing 2.4, you saw a straightforward example of using <section> within an <article> (repeated here). In this example, you can easily

imagine that the <section>s contain a "thematic grouping of content," which is supported by the fact that they each have a heading:

Click here to view code image

```
<article>
    <header>
       <h2>Article Heading</h2>
    </header>
    <section>
       <header>
          <h3>Section 1 Heading</h3>
       </header>
       <p>Section 1 content here.</p>
    </section>
    <section>
       <header>
          <h3>Section 2 Heading</h3>
       </header>
       <p>Section 2 content here.</p>
    </section>
    <footer>
       <p>Article footer goes here.</p>
    </footer>
</article>
```

But here's an example of a perfectly valid use of <section> with no <article> element in sight:

Click here to view code image

```
<section>
    <header>
       <h1>Super Heading</h1>
    </header>
    <p>Super content!</p>
</section>
```

So what's a developer to do? Let's say you have some generic content that you know you want to divide into sections with their own headings. In that case, use <section>. If you need to only *visually* delineate chunks of content (such as with paragraph breaks) that do not require additional headings, then <section> isn't for you—use <p> or instead.

Because the <section> element can contain any other flow content element, and can be contained within any other flow content element

(except the <address> element, discussed later in this chapter), it's easy to see why, without other limitations and with generic guidelines for use, the <section> element is sometimes misunderstood.

Using <article> Appropriately

Personally, I believe that a lot of the misunderstanding regarding the use of <section> versus <article> has to do with the name of the <article> element. When I think of an article, I think specifically about an article in a newspaper or a magazine. I don't naturally think "any standalone body of work," which is how the <article> element is commonly defined. The HTML5 recommended specification defines it as "a complete, or self-contained, composition in a document, page, application, or site and that is, in principle, independently distributable or reusable," such as "a forum post, a magazine or newspaper article, a blog entry, a user-submitted comment, an interactive widget or gadget, or any other independent item of content."

In other words, an <article> element could be used to contain the entire page of a website (whether or not it is an article in a publication), an actual article in a publication, a blog post anywhere and everywhere, part of a threaded discussion in a forum, a comment on a blog post, and as a container that displays the current weather in your city. It's no wonder there are tens of thousands of results for a search on "difference between section and article in HTML5."

A good rule of thumb when you're trying to figure out when to use <article> and when to use <section> is simply to answer the following question: Does this content make sense on its own? If so, then no matter what the content seems to be to you (for example, a static web page, not an article in the *New York Times*), start by using the <article> element. If you find yourself breaking it up, do so in <section>s. And if you find yourself thinking that your "article" is, in fact, part of a greater whole, then change the <article> tags to <section> tags, find the beginning of the document, and surround it from there with the more appropriately placed <article> tag at a higher level.

Implementing the <nav> Element

The <nav> element seems so simple (<nav> implies *navigation*), and it ultimately is—but it can also be used incorrectly. In this section, you'll learn some basic uses, and also some incorrect uses to avoid. If your site has any navigational elements at all, either sitewide or within a long page of content, you have a valid use for the <nav> element.

For that sitewide navigation, you typically find a <nav> element within the primary <header> element; you are not required to put it there, but if you want your navigational content to be introductory (and omnipresent in your template), you can easily make a case for your primary <nav> element to appear within the primary <header>. More important, that is valid HTML (as is <nav> outside a <header>) because a <nav> element can appear within any flow content, as well as contain any flow content.

The following code snippet shows the main navigational links of a website, placed within a <header> element:

Click here to view code image

```
<header>
   <img src="acmewidgets.jpg" alt="ACME Widgets LLC"/>
   <h1>The finest widgets are made here!</h1>
   <nav>
      <ul>
         <li><a href="#">About Us</a></li>
         <li><a href="#">Products</a></li>
         <li><a href="#">Support</a></li>
         <li><a href="#">Press</a></li>
      </ul>
   </nav>
</header>
```

You are not limited to a single <nav> element in your documents, which is good for site developers who create templates that include both primary *and* secondary navigation. For example, you might see horizontal primary navigation at the top of a page (often contained within a <header> element), and then vertical navigation in the left column of a page, representing the secondary pages within the main section. In that case, you simply use a second <nav> element, not contained within the <header>,

placed and styled differently to delineate the two types visually in addition to semantically.

Remember, the `<nav>` element is used for *major* navigational content—primary and secondary navigation both count, as does the inclusion of tables of contents within a page. For good and useful semantic use of the `<nav>` element, do not simply apply it to every link that allows a user to navigate anywhere. Note that I said "good and useful" semantic use, not necessarily "valid" use—it's true that you could apply `<nav>` to any list of links, and it would be valid according to the HTML specification because links are flow content. But it wouldn't be particularly *useful*—it wouldn't add meaning—to surround a list of links to social media sharing tools with the `<nav>` element.

When to Use `<aside>`

As you'll see by the number of tips and notes from me throughout this book, I'm a big fan of the type of content that is most appropriately marked up within the `<aside>` element. The `<aside>` element is meant to contain any content that is tangentially related to the content around it—additional explanation, links to related resources, pull quotes, helper text, and so on. You might think of the `<aside>` element as a sidebar, but be careful not to think of it only as a *visual* sidebar, or a column on the side of a page where you can stick anything and everything you want, whether or not it's related to the content or site at hand.

In Figure 2.8, you can see how content in an `<aside>` is used to create a *pull quote*, or a content excerpt that is specifically set aside to call attention to it. The `<aside>`, in this case, is used to highlight an important section of the text, but it could also have been used to define a term or link to related documents.

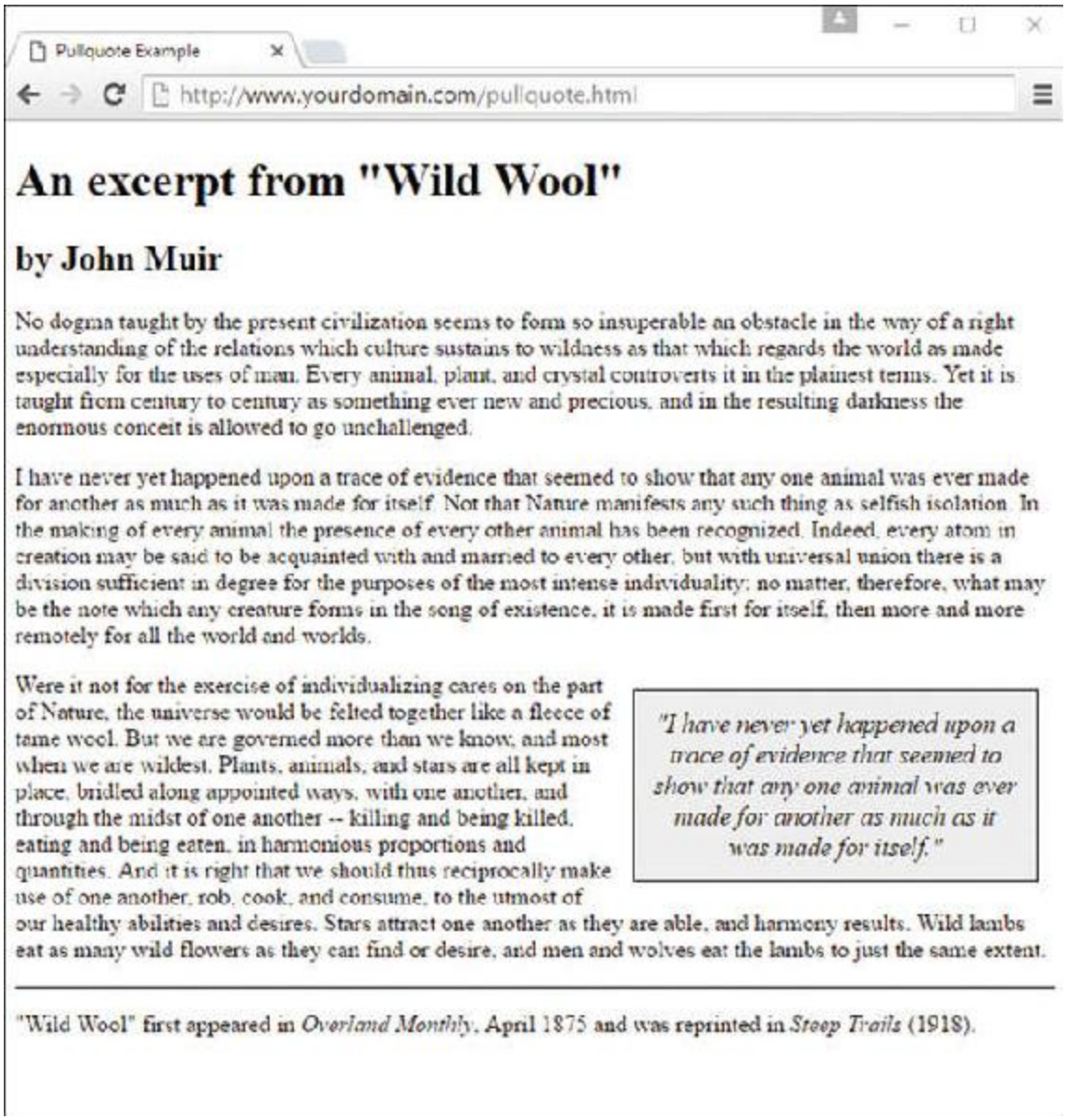

FIGURE 2.8
Using `<aside>` to create meaningful pull quotes.

When determining whether to use the `<aside>` element, think about the content you want to add. Is it related directly to the content in which the `<aside>` would be contained, such as a definition of terms used in an article or a list of related links for the article? If your answer is an easy yes, that's great! Use `<aside>` to your heart's content. If you're thinking of including an `<aside>` outside a containing element that is itself full of content, just make sure that the content of the `<aside>` is reasonably

related to your site overall and that you're not just using the <aside> element for visual effect.

Using <footer> Effectively

The counterpart to the <header> element, the <footer> element, contains additional information about its containing element. The most common use of the <footer> element is to contain copyright information at the bottom of a page, such as here:

Click here to view code image

```
<footer>
   <p>&copy; 2017 Acme Widgets, LLC. All Rights Reserved.</p>
</footer>
```

Similar to the <header> element, the <footer> element can contain any other element in the flow content category, of which it is also a member, with the exception of *other* <footer> or <header> elements. Additionally, a <footer> element cannot be contained within an <address> element, but a <footer> element can contain an <address> element—in fact, a <footer> element is a common location for an <address> element to reside in.

Placing useful <address> content within a <footer> element is one of the most effective uses of the <footer> element (not to mention the <address> element) because it provides specific contextual information about the page or section of the page to which it refers. The following snippet shows a use of <address> within <footer>:

Click here to view code image

```
<footer>
   <p>&copy; 2017 Acme Widgets, LLC. All Rights Reserved.</p>
   <p>Copyright Issues? Contact:</p>
      <address>
      Our Lawyer<br>
      123 Main Street<br>
      Somewhere, CA 95128<br>
      <a href="mailto:lawyer@example.com">lawyer@example.com</a>
      </address>
</footer>
```

As with the <header> element, you are not limited to only one <footer>. You can use as many <footer> elements as you need, as long as they are containers for additional information about the containing element—Listing 2.4 showed the use of <footer> elements for both a page and an <article>, both of which are valid.

How CSS Works

In the preceding sections, you learned the basics of HTML, including how to set up a skeletal HTML template for all your web content, use hyperlinks, and generally organize your content. In this section, you'll learn the basics of fine-tuning the visual display of your web content using *Cascading Style Sheets (CSS)*.

The concept behind style sheets is simple: You create a style sheet document that specifies the fonts, colors, spacing, and other characteristics that establish a unique look for a website. You then link every page that should have that look to the style sheet instead of specifying all those styles repeatedly in each separate document. Therefore, when you decide to change your official corporate typeface or color scheme, you can modify all your web pages at once just by changing one or two entries in your style sheet—you don't have to change them in all your static web files. So a *style sheet* is a grouping of formatting instructions that control the appearance of several HTML pages at once.

Style sheets enable you to set a great number of formatting characteristics, including exact typeface controls, letter and line spacing, and margins and page borders, just to name a few. Style sheets also enable you to specify sizes and other measurements in familiar units, such as inches, millimeters, points, and picas. In addition, you can use style sheets to precisely position graphics and text anywhere on a web page, either at specific coordinates or relative to other items on the page.

In short, style sheets bring a sophisticated level of display to the Web—and they do so, if you'll pardon the expression, with style.

NOTE

If you have three or more web pages that share (or should share) similar formatting and fonts, you might want to create a style sheet for them as you read this chapter. Even if you choose not to create a complete style sheet, you'll find it helpful to apply styles to individual HTML elements directly within a web page.

A *style rule* is a formatting instruction that can be applied to an element on a web page, such as a paragraph of text or a link. Style rules consist of one or more style properties and their associated values. An *internal style sheet* is placed directly within a web page, whereas an *external style sheet* exists in a separate document and is simply linked to a web page via a special tag —more on this tag in a moment.

The *cascading* part of the name Cascading Style Sheets refers to the manner in which style sheet rules are applied to elements in an HTML document. More specifically, styles in a CSS style sheet form a hierarchy in which more specific styles override more general styles. It is the responsibility of CSS to determine the precedence of style rules according to this hierarchy, which establishes a cascading effect. If that sounds a bit confusing, just think of the cascading mechanism in CSS as being similar to genetic inheritance, in which general traits are passed from parents to a child, but more specific traits are entirely unique to the child. Base-style rules are applied throughout a style sheet but can be overridden by more specific style rules.

NOTE

You might notice that I use the term *element* a fair amount in this chapter (and I do in the rest of the book, for that matter). An *element* is simply a piece of information (content) in a web page, such as an image, a paragraph, or a link. Tags are used to mark up elements, and you can think of an element as a tag, complete with descriptive information (attributes, text, images, and so on) within the tag.

A quick example should clear things up. Take a look at the following code to see whether you can tell what's going on with the color of the text:

Click here to view code image

```
<div style="color:green">
  This text is green.
  <p style="color:blue">This text is blue.</p>
  <p>This text is still green.</p>
</div>
```

In the preceding example, the color green is applied to the `<div>` tag via the `color` style property. Therefore, the text in the `<div>` tag is colored green. Because both `<p>` tags are children of the `<div>` tag, the green text style cascades down to them. However, the first `<p>` tag overrides the color style and changes it to blue. The end result is that the first line (not surrounded by a paragraph tag) is green, the first official paragraph is blue, and the second official paragraph retains the cascaded green color.

If you made it through that description on your own and came out on the other end unscathed, congratulations—that's half the battle. Understanding CSS isn't like understanding rocket science—the more you practice, the more it will become clear. The real trick is developing the aesthetic design sense that you can then apply to your online presence through CSS.

Like many web technologies, CSS has evolved over the years. The original version of CSS, known as *Cascading Style Sheets Level 1* (*CSS1*), was created in 1996. The later CSS2 standard was created in 1998, and CSS2 is still in use today; all modern web browsers support CSS2. The latest version of CSS is CSS3, which builds on the strong foundation laid by its predecessors but adds advanced functionality to enhance the online experience. In the following sections you'll learn core CSS, and throughout the rest of the book when CSS is discussed or used, I'll refer to CSS3.

The rest of this chapter explains the basics of putting CSS to good use, but it's not a reference for all things CSS. Nor is the rest of this book, which will show plenty of examples of basic CSS in use as you learn to build dynamic web applications. However, you can find a developer-oriented guide to CSS at https://developer.mozilla.org/en-US/docs/Web/CSS that gets into excruciating detail regarding everything you can do with CSS.

This guide can be an invaluable reference to you as you continue on your web development journey.

A Basic Style Sheet

Despite their power, style sheets are simple to create. Consider the web pages shown in Figures 2.9 and 2.10. These pages share several visual properties that can be put into a common style sheet:

- They use a large, bold Verdana font for the headings and a normal-size and normal-weight Verdana font for the body text.
- They use an image named `logo.gif` floating within the content and on the right side of the page.
- All text is black except for subheadings, which are purple.
- They have margins on the left side and at the top.
- They include vertical space between lines of text.
- They include a footer that is centered and in small print.

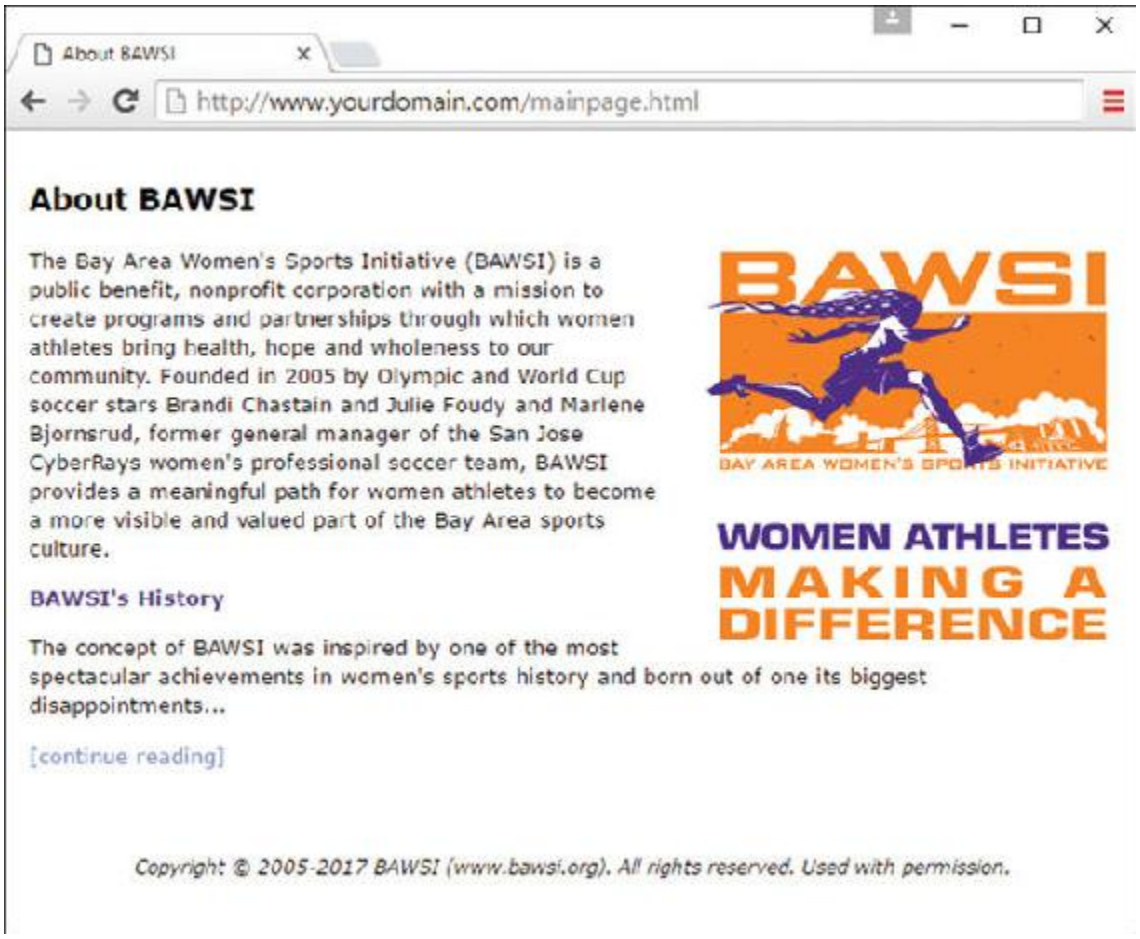

FIGURE 2.9
This page uses a style sheet to fine-tune the appearance and spacing of the text and images.

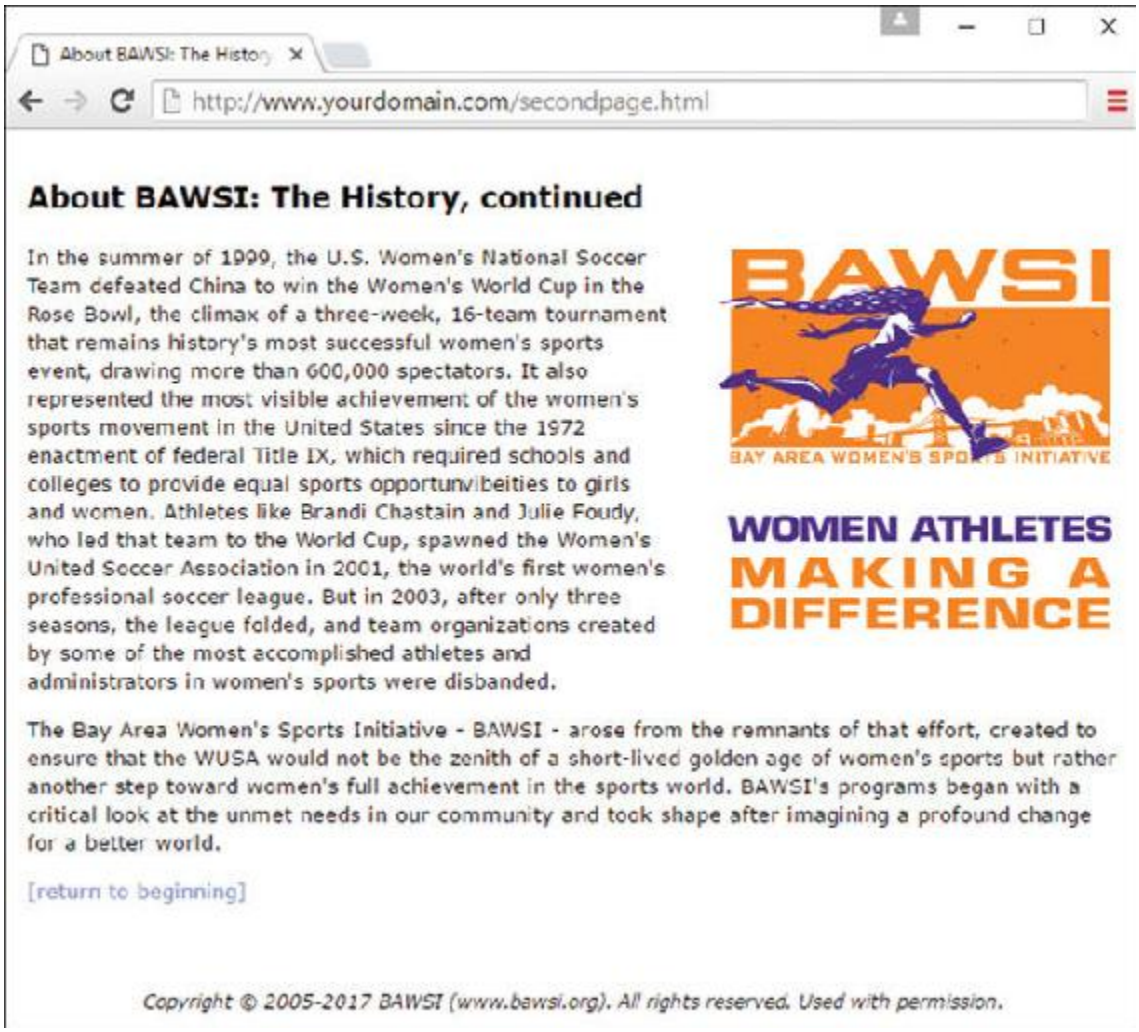

FIGURE 2.10
This page uses the same style sheet as the one in Figure 2.9, thus maintaining a consistent look and feel.

Listing 2.5 shows the CSS used in a style sheet to specify these properties.

LISTING 2.5 A Single External Style Sheet

Click here to view code image

```
body {
  font-size: 10pt;
  font-family: Verdana, Geneva, Arial, Helvetica, sans-serif;
  color: black;
  line-height: 14pt;
  padding-left: 5pt;
  padding-right: 5pt;
  padding-top: 5pt;
```

```
}

h1 {
  font: 14pt Verdana, Geneva, Arial, Helvetica, sans-serif;
  font-weight: bold;
  line-height: 20pt;
}

p.subheader {
  font-weight: bold;
  color: #593d87;
}

img {
  padding: 3pt;
  float: right;
}

a {
  text-decoration: none;
}

a:link, a:visited {
  color: #8094d6;
}

a:hover, a:active {
  color: #FF9933;
}

footer {
  font-size: 9pt;
  font-style: italic;
  line-height: 12pt;
  text-align: center;
  padding-top: 30pt;
}
```

This might initially appear to be a lot of code, but if you look closely, you'll see that there isn't a lot of information on each line of code. It's fairly standard to place individual style rules on their own line, to help make style sheets more readable, but that is a personal preference; you could put all the rules on one line as long as you kept using the semicolon to separate each rule (more on that in a bit). Speaking of code readability, perhaps the first thing you noticed about this style sheet code is that it doesn't look anything

like normal HTML code. CSS uses a syntax all its own to specify style sheets.

Of course, the listing includes some familiar HTML tags (although not all tags require an entry in the style sheet). As you might guess, `body`, `h1`, `p`, `img`, `a`, and `footer` in the style sheet refer to the corresponding tags in the HTML documents to which the style sheet will be applied. The curly braces after each tag name describe how all content within that tag should appear.

In this case, the style sheet says that all body text should be rendered at a size of 10 points, in the Verdana font (if possible), and with the color black, with 14 points between lines. If the user does not have the Verdana font installed, the list of fonts in the style sheet represents the order in which the browser should search for fonts to use: Geneva, then Arial, and then Helvetica. If the user has none of those fonts, the browser uses whatever default sans-serif font is available. Additionally, the page should have left, right, and top padding of 5 points each.

Any text within an `<h1>` tag should be rendered in boldface Verdana at a size of 14 points. Moving on, any paragraph that uses only the `<p>` tag inherits all the styles indicated by the body element. However, if the `<p>` tag uses a special class named `subheader`, the text appears bold and in the color `#593d87` (a purple color).

The `pt` after each measurement in Listing 2.5 means *points* (there are 72 points in an inch). If you prefer, you can specify any style sheet measurement in inches (`in`), centimeters (`cm`), pixels (`px`), or "widths of a letter *m*," which are called ems (`em`).

You might have noticed that each style rule in the listing ends with a semicolon (`;`). Semicolons are used to separate style rules from each other. It is therefore customary to end each style rule with a semicolon so that you can easily add another style rule after it. Review the remainder of the style sheet in Listing 2.5 to see the presentation formatting applied to additional tags.

NOTE

You can specify font sizes as large as you like with style sheets, although some display devices and printers do not correctly handle fonts larger than 200 points.

To link this style sheet to HTML documents, include a `<link>` tag in the `<head>` section of each document. Listing 2.6 shows the HTML code for the page shown in Figure 2.9. It contains the following `<link>` tag:

Click here to view code image

```
<link rel="stylesheet" type="text/css" href="styles.css">
```

This assumes that the style sheet is stored under the name `styles.css` in the same folder as the HTML document. As long as the web browser supports style sheets—and all modern browsers do—the properties specified in the style sheet will apply to the content in the page without the need for any special HTML formatting code. This meets one of the goals of HTML, which is to provide a separation between the content in a web page and the specific formatting required to display that content.

LISTING 2.6 HTML Code for the Page Shown in Figure 2.9

Click here to view code image

```
<!DOCTYPE html>

<html lang="en">
  <head>
    <title>About BAWSI</title>
    <link rel="stylesheet" type="text/css" href="styles.css">
  </head>
  <body>
     <section>

     <header>
     <h1>About BAWSI</h1>
     </header>

     <p><img src="logo.gif" alt="BAWSI logo">The Bay Area Women's
     Sports Initiative (BAWSI) is a public benefit, nonprofit
     corporation with a mission to create programs and partnerships
     through which women athletes bring health, hope and wholeness
```

```
to
      our community. Founded in 2005 by Olympic and World Cup soccer
      stars Brandi Chastain and Julie Foudy and Marlene Bjornsrud,
      former general manager of the San Jose CyberRays women's
      professional soccer team, BAWSI provides a meaningful path for
      women athletes to become a more visible and valued part of the
      Bay Area sports culture.</p>

      <p class="subheader">BAWSI's History</p>

      <p>The concept of BAWSI was inspired by one of the most
      spectacular achievements in women's sports history and born
out
      of one its biggest disappointments… </p>

      <p><a href="secondpage.html">[continue reading]</a></p>
      </section>


      <footer>
      Copyright &copy; 2005-2017 BAWSI (www.bawsi.org).
      All rights reserved.  Used with permission.
      </footer>
  </body>
</html>
```

TIP

In most web browsers, you can view the style rules in a style sheet by opening the `.css` file and choosing Notepad or another text editor as the helper application to view the file. (To determine the name of the `.css` file, look at the HTML source of any web page that links to it.) To edit your own style sheets, just use a text editor.

The code in Listing 2.6 is interesting because it contains no formatting of any kind. In other words, nothing in the HTML code dictates how the text and images are to be displayed—no colors, no fonts, nothing. Yet the page is carefully formatted and rendered to the screen, thanks to the link to the external style sheet, `styles.css`. The real benefit to this approach is that you can easily create a site with multiple pages that maintains a consistent look and feel. And you have the benefit of isolating the visual style of the

page to a single document (the style sheet) so that one change impacts all pages.

NOTE

Not every browser's support of CSS is flawless. To find out how major browsers compare to each other in terms of CSS support, take a look at these websites: http://www.quirksmode.org/css/contents.html and http://caniuse.com.

▼ TRY IT YOURSELF

Create a Style Sheet of Your Own

Starting from scratch, create a new text document called `mystyles.css` and add some style rules for the following basic HTML tags: `<body>`, `<p>`, `<h1>`, and `<h2>`. After creating your style sheet, make a new HTML file that contains these basic tags. Play around with different style rules and see for yourself how simple it is to change entire blocks of text in paragraphs with one simple change in a style sheet file.

A CSS Style Primer

You now have a basic knowledge of CSS style sheets and how they are based on style rules that describe the appearance of information in web pages. The next few sections of this chapter provide a quick overview of some of the most important style properties and enable you to get started using CSS in your own style sheets.

CSS includes various style properties that are used to control fonts, colors, alignment, and margins, to name just a few. The style properties in CSS can be generally grouped into two major categories:

- Layout properties, which consist of properties that affect the positioning of elements on a web page, such as margins, padding, and alignment
- Formatting properties, which consist of properties that affect the visual display of elements within a website, such as the font type, size, and color

Basic Layout Properties

CSS layout properties determine how content is placed on a web page. One of the most important layout properties is the `display` property, which describes how an element is displayed with respect to other elements. The `display` property has four basic values:

- **`block`**—The element is displayed on a new line, as in a new paragraph.
- **`list-item`**—The element is displayed on a new line with a list-item mark (bullet) next to it.
- **`inline`**—The element is displayed inline with the current paragraph.
- **`none`**—The element is not displayed; it is hidden.

NOTE

The `display` property relies on a concept known as *relative positioning*, which means that elements are positioned relative to the location of other elements on a page. CSS also supports *absolute positioning*, which enables you to place an element at an exact location on a page, independent of other elements. You'll learn more about both of these types of positioning in Chapter 3, "Understanding the CSS Box Model and Positioning."

Understanding the `display` property is easier if you visualize each element on a web page occupying a rectangular area when displayed—the `display` property controls the manner in which this rectangular area is displayed. For example, the `block` value results in the element being placed on a new line by itself, whereas the `inline` value places the

element next to the content just before it. The `display` property is one of the few style properties that can be applied in most style rules. Following is an example of how to set the `display` property:

```
display: block;
```

You control the size of the rectangular area for an element with the `width` and `height` properties. As with many size-related CSS properties, `width` and `height` property values can be specified in several different units of measurement:

- **`in`**—Inches
- **`cm`**—Centimeters
- **`mm`**—Millimeters
- **`%`**—Percentage
- **`px`**—Pixels
- **`pt`**—Points

You can mix and match units however you choose within a style sheet, but it's generally a good idea to be consistent across a set of similar style properties. For example, you might want to stick with points for font properties and pixels for dimensions. Following is an example of setting the width of an element using pixel units:

```
width: 200px;
```

Basic Formatting Properties

CSS formatting properties are used to control the appearance of content on a web page, as opposed to controlling the physical positioning of the content. One of the most popular formatting properties is the `border` property, which establishes a visible boundary around an element with a box or partial box. Note that a border is always present in that space is always left for it, but the border does not appear in a way that you can see unless you give it properties that make it visible (like a color). The

following border properties provide a means of describing the borders of an element:

- **`border-width`**—The width of the border edge
- **`border-color`**—The color of the border edge
- **`border-style`**—The style of the border edge
- **`border-left`**—The left side of the border
- **`border-right`**—The right side of the border
- **`border-top`**—The top of the border
- **`border-bottom`**—The bottom of the border
- **`border`**—All the border sides

The `border-width` property establishes the width of the border edge. It is often expressed in pixels, as the following code demonstrates:

```
border-width: 5px;
```

Not surprisingly, the `border-color` and `border-style` properties set the border color and style. Following is an example of how these two properties are set:

```
border-color: blue;
border-style: dotted;
```

The `border-style` property can be set to any of the following basic values:

- **`solid`**—A single-line border
- **`double`**—A double-line border
- **`dashed`**—A dashed border
- **`dotted`**—A dotted border
- **`groove`**—A border with a groove appearance
- **`ridge`**—A border with a ridge appearance
- **`inset`**—A border with an inset appearance

- **`outset`**—A border with an outset appearance
- **`none`**—No border
- **`hidden`**—Effectively the same as `none` in that no border is displayed, but if two elements are next to each other with collapsed space between them, `hidden` ensures that a collapsed visible border does not show within the area of the element with a hidden border.

The default value of the `border-style` property is `none`, which is why elements don't have a border *unless* you set the `border` property to a different style. Although `solid` is the most common border style, you will also see the other styles in use.

The `border-left`, `border-right`, `border-top`, and `border-bottom` properties enable you to set the border for each side of an element individually. If you want a border to appear the same on all four sides, you can use the single `border` property by itself, which expects the following styles separated by a space: `border-width`, `border-style`, and `border-color`. Following is an example of using the `border` property to set a border that consists of two (double) red lines that are a total of 10 pixels in width:

```
border: 10px double red;
```

Whereas the color of an element's border is set with the `border-color` property, the color of the inner region of an element is set using the `color` and `background-color` properties. The `color` property sets the color of text in an element (foreground), and the `background-color` property sets the color of the background behind the text. Following is an example of setting both color properties to predefined colors:

```
color: black;
background-color: orange;
```

You can also assign custom colors to these properties by specifying the colors in hexadecimal or as RGB (Red, Green, Blue) decimal values:

```
background-color: #999999;
color: rgb(0,0,255);
```

You can also control the alignment and indentation of web page content without too much trouble. This is accomplished with the `text-align` and `text-indent` properties, as the following code demonstrates:

```
text-align: center;
text-indent: 12px;
```

When you have an element properly aligned and indented, you might be interested in setting its font. The following basic font properties set the various parameters associated with fonts:

- **`font-family`**—The family of the font
- **`font-size`**—The size of the font
- **`font-style`**—The style of the font (normal or italic)
- **`font-weight`**—The weight of the font (normal, lighter, bold, bolder, and so on)

The `font-family` property specifies a prioritized list of font family names. A prioritized list is used instead of a single value to provide alternatives in case a font isn't available on a given system. The `font-size` property specifies the size of the font using a unit of measurement, often in points. Finally, the `font-style` property sets the style of the font, and the `font-weight` property sets the weight of the font. Following is an example of setting these font properties:

Click here to view code image

```
font-family: Arial, sans-serif;
font-size: 36pt;
font-style: italic;
font-weight: normal;
```

Now that you know a whole lot more about style properties and how they work, refer to Listing 2.5 and see whether it makes a bit more sense. Here's a recap of the style properties used in that style sheet, which you can use as a guide for understanding how it works:

- **`font`**—Lets you set many font properties at once. You can specify a list of font names separated by commas; if the first is not available, the

next is tried, and so on. You can also include the words bold and/or italic and a font size. Alternatively, you can set each of these `font` properties separately with `font-family`, `font-size`, `font-weight`, and `font-style`.

- **`line-height`**—Also known in the publishing world as *leading*. This sets the height of each line of text, usually in points.
- **`color`**—Sets the text color using the standard color names or hexadecimal color codes.
- **`text-decoration`**—Is useful for turning off link underlining; simply set it to `none`. The values of `underline`, `italic`, and `line-through` are also supported.
- **`text-align`**—Aligns text to the left, right, or center, along with justifying the text with a value of `justify`.
- **`padding`**—Adds padding to the left, right, top, and bottom of an element; this padding can be in measurement units or a percentage of the page width. Use `padding-left` and `padding-right` if you want to add padding to the left and right of the element independently. Use `padding-top` or `padding-bottom` to add padding to the top or bottom of the element, as appropriate. You'll learn a bit more about these style properties in Chapter 3.

Using Style Classes

Whenever you want some of the text on your pages to look different from the other text, you can create what amounts to a custom-built HTML tag. Each type of specially formatted text you define is called a *style class*, which is a custom set of formatting specifications that can be applied to any element in a web page.

Before showing you a style class, I need to take a quick step back and clarify some CSS terminology. First off, a CSS *style property* is a specific style that you can assign a value, such as `color` or `font-size`. You associate a style property and its respective value with elements on a web page by using a selector. A *selector* is used to identify tags on a page to

which you apply styles. Following is an example of a selector, a property, and a value all included in a basic style rule:

```
h1 { font: 36pt Courier; }
```

In this code, `h1` is the selector, `font` is the style property, and `36pt Courier` is the value. The selector is important because it means that the font setting will be applied to all `h1` elements in the web page. But maybe you want to differentiate between some of the `h1` elements—what then? The answer lies in style classes.

Suppose you want two different kinds of `<h1>` headings for use in your documents. You create a style class for each one by putting the following CSS code in a style sheet:

Click here to view code image

```
h1.silly { font: 36pt 'Comic Sans'; }
h1.serious { font: 36pt Arial; }
```

Notice that these selectors include a period (`.`) after `h1`, followed by a descriptive class name. To choose between the two style classes, use the `class` attribute, like this:

Click here to view code image

```
<h1 class="silly">Marvin's Munchies Inc. </h1>
<p>Text about Marvin's Munchies goes here. </p>
```

Or you could use this:

Click here to view code image

```
<h1 class="serious">MMI Investor Information</h1>
<p>Text for business investors goes here.</p>
```

When referencing a style class in HTML code, simply specify the class name in the `class` attribute of an element. In the preceding example, the words `Marvin's Munchies Inc.` would appear in a 36-point Comic Sans font, assuming that you included a `<link>` to the style sheet at the top of the web page and that the user has the Comic Sans font installed. The words `MMI Investor Information` would appear in the 36-point

Arial font instead. You can see another example of classes in action in Listing 2.5; look for the `specialtext` `<p>` class.

What if you want to create a style class that can be applied to any element instead of just headings or some other particular tag? In your CSS, simply use a period (`.`) followed by any style class name you make up and any style specifications you choose. That class can specify any number of font, spacing, and margin settings all at once. Wherever you want to apply your custom tag in a page, just use an HTML tag plus the `class` attribute, followed by the class name you created.

For example, the style sheet in Listing 2.5 includes the following style class specification:

```
p.specialtext {
  font-weight: bold;
  color: #593d87;
}
```

This style class is applied in Listing 2.6 with the following tag:

```
<p class="specialtext">
```

TIP

You might have noticed a change in the coding style when a style rule includes multiple properties. For style rules with a single style, you'll commonly see the property placed on the same line as the rule, like this:

Click here to view code image

```
p.specialtext { font-weight: bold; }
```

However, when a style rule contains multiple style properties, it's much easier to read and understand the code if you list the properties one per line, like this:

```
p.specialtext {
  font-weight: bold;
  color: #593d87;
}
```

Everything between that tag and the closing tag in Listing 2.6 appears in bold purple text.

If no element selector were present in your style sheet, meaning that the rule looked like this:

```
.specialtext {
  font-weight: bold;
  color: #593d87;
}
```

Then any element could refer to `specialtext` and have the text rendered as bold purple, not just a element.

What makes style classes so valuable is how they isolate style code from web pages, effectively enabling you to focus your HTML code on the actual content in a page, not on how it is going to appear on the screen. Then you can focus on how the content is rendered to the screen by fine-tuning the style sheet. You might be surprised by how a relatively small amount of code in a style sheet can have significant effects across an entire website. This makes your pages much easier to maintain and manipulate.

Using Style IDs

When you create custom style classes, you can use those classes as many times as you like—they are not unique. However, in some instances, you want precise control over unique elements for layout or formatting purposes (or both). In such instances, look to ID*s* instead of classes.

A *style ID* is a custom set of formatting specifications that can be applied to only one element in a web page. You can use IDs across a set of pages, but only once per time within each page.

For example, suppose you have a title within the body of all your pages. Each page has only one title, but all the pages themselves include one instance of that title. Following is an example of a selector with an ID indicated, plus a property and a value:

Click here to view code image

```
p#title {font: 24pt Verdana, Geneva, Arial, sans-serif}
```

Notice that this selector includes a hash mark, or pound sign (#), after p, followed by a descriptive ID name. When referencing a style ID in HTML code, simply specify the ID name in the id attribute of an element, like so:

Click here to view code image

```
<p id="title">Some Title Goes Here</p>
```

Everything between the opening and closing <p> tags will appear in 24-point Verdana text—but only once on any given page. You often see style IDs used to define specific parts of a page for layout purposes, such as a header area, footer area, main body area, and so on. These types of areas in a page appear only once per page, so using an ID rather than a class is the appropriate choice.

Internal Style Sheets and Inline Styles

In some situations, you want to specify styles that will be used in only one web page. You can enclose a style sheet between <style> and </style> tags and include it directly in an HTML document. Style sheets used in this manner must appear in the <head> of an HTML document. No <link> tag is needed, and you cannot refer to that style sheet from any other page (unless you copy it into the beginning of that document too). This kind of style sheet is known as an internal style sheet, as you learned earlier in the chapter.

Listing 2.7 shows an example of how you might specify an internal style sheet.

LISTING 2.7 A Web Page with an Internal Style Sheet

Click here to view code image

```
<!DOCTYPE html>

<html lang="en">
  <head>
```

```
    <title>Some Page</title>

    <style type="text/css">
      footer {
        font-size: 9pt;
        line-height: 12pt;
        text-align: center;
      }
    </style>
  </head>
  <body>
  …
    <footer>
    Copyright 2017 Acme Products, Inc.
    </footer>
  </body>
</html>
```

In the listing code, the `footer` style class is specified in an internal style sheet that appears in the head of the page. The style class is now available for use within the body of this page, and only within this page. In fact, it is used in the body of the page to style the copyright notice.

Internal style sheets are handy if you want to create a style rule that is used multiple times within a single page. However, in some instances, you might need to apply a unique style to one particular element. This calls for an inline style rule, which enables you to specify a style for only a small part of a page, such as an individual element. For example, you can create and apply a style rule within a `<p>`, `<div>`, or `<span>` tag via the `style` attribute. This type of style is known as an *inline style* because it is specified right there in the middle of the HTML code.

NOTE

The `<span>` and `</span>` tags are dummy tags that do nothing in and of themselves except specify a range of content to apply any `style` attributes that you add. The only difference between `<div>` and `<span>` is that `<div>` is a block element and, therefore, forces a line break, whereas `<span>` is an inline element and doesn’t force a break. Therefore, you

should use to modify the style of any portion of text that is to appear in the middle of a sentence or paragraph without any line break.

Here's how a sample style attribute might look:

Click here to view code image

```
<p style="color:green">
  This text is green, but <span style="color:red">this text is
  red.</span>
  Back to green again, but…
</p>
<p>
  …now the green is over, and we're back to the default color
  for this page.
</p>
```

This code makes use of the tag to show how to apply the color style property in an inline style rule. In fact, both the <p> tag and the tag in this example use the color property as an inline style. What's important to understand is that the color:red style property overrides the color:green style property for the text between the and tags. Then, in the second paragraph, neither of the color styles applies because it is a completely new paragraph that adheres to the default color of the entire page.

CAUTION

Using inline styles isn't considered a best practice when used beyond page-level debugging or beyond trying out new things in a controlled setting. The best practice of all is having your pages link to a centrally maintained style sheet so that changes are immediately reflected in all pages that use it.

Validate Your Style Sheets

Just as it is important to validate your HTML markup, it is important to validate your style sheet. You can find a specific validation tool for CSS at http://jigsaw.w3.org/css-validator/. You can point the tool to a web address,

upload a file, or paste content into the form field provided. The ultimate goal is a result like the one in Figure 2.11: `valid`!

FIGURE 2.11
The W3C CSS Validator shows there are no errors in the style sheet contents of Listing 2.5.

Summary

This chapter introduced the basics of what web pages are and how they work. You learned that coded HTML commands are included in a text file, and you saw that typing HTML text yourself is better than using a graphical editor to create HTML commands for you—especially when you're learning HTML.

You were introduced to the most basic and important HTML tags. By adding these coded commands to any plain-text document, you can quickly transform it into a bona fide web page. You learned that the first step in creating a web page is to put a few obligatory HTML tags at the beginning and end, including adding a title for the page. You can then mark where paragraphs and lines end and add horizontal rules and headings, if you want them. You also got a taste of some of the semantic tags in HTML5, which are used to provide additional meaning by delineating the types of content your pages contain (not just the content itself). Table 2.1 summarizes the basic HTML tags introduced in this chapter.

Beyond HTML, you learned that a style sheet can control the appearance of many HTML pages at once. It can also give you extremely precise control over the typography, spacing, and positioning of HTML elements. You also learned that, by adding a `style` attribute to almost any HTML tag, you can control the style of any part of an HTML page without referring to a separate style sheet document.

You learned about three main approaches to including style sheets in your website: a separate style sheet file with the extension `.css` that is linked to in the `<head>` of your documents, a collection of style rules placed in the head of the document within the `<style>` tag, and rules placed directly in an HTML tag via the `style` attribute (although the latter is not a best practice for long-term use). Table 2.2 summarizes tags with attributes discussed in this chapter.

TABLE 2.1 HTML Tags Covered in this Chapter

Tag	Function
`<html>...</html>`	Encloses the entire HTML document.
`<head>...</head>`	Encloses the head of the HTML document. Used within the `<html>` tag pair.
`<title>...</title>`	Indicates the title of the document. Used within the `<head>` tag pair.

`<body>…</body>`	Encloses the body of the HTML document. Used within the `<html>` tag pair.
`<p>…</p>`	Encloses a paragraph; skips a line between paragraphs.
` `	Indicates a line break.
`<hr>`	Displays a horizontal rule line.
`<h1>…</h1>`	Encloses a first-level heading.
`<h2>…</h2>`	Encloses a second-level heading.
`<h3>…</h3>`	Encloses a third-level heading.
`<h4>…</h4>`	Encloses a fourth-level heading.
`<h5>…</h5>`	Encloses a fifth-level heading.
`<h6>…</h6>`	Encloses a sixth-level heading.
`<header>…</header>`	Contains introductory information.
`<footer>…</footer>`	Contains supplementary material for its containing element (commonly a copyright notice or author information).
`<nav>…</nav>`	Contains navigational elements.
`<section>…</section>`	Contains thematically similar content, such as a chapter of a book or a section of a page.
`<article>…</article>`	Contains content that is a standalone body of work, such as a news article.
`<aside>…</aside>`	Contains secondary information for its containing element.
`<address>…</address>`	Contains address information related to its nearest `<article>` or `<body>` element, often contained within a `<footer>` element.

TABLE 2.2 HTML Tags with Attributes Covered in This Chapter

Tag/Attributes	Function
Tag	
`<a>`	Indicates a hyperlink to a position in the current document or to another document.
Attributes	
`href="url"`	The address of the linked content.
Tag	
`<style>…</style>`	Allows an internal style sheet to be included within a document. Used between `<head>` and `</head>`.
Attribute	
`type="contenttype"`	The Internet content type. (Always `"text/css"` for a CSS style sheet.)
Tag	
`<link>`	Links to an external style sheet (or other document type). Used in the `<head>` section of the document.
Attributes	
`href="url"`	The address of the style sheet.
`type="contenttype"`	The Internet content type. (Always `"text/css"` for a CSS style sheet.)
`rel="stylesheet"`	The relationship to a referenced document. (Always `"stylesheet"` for style sheets.)
Tag	
`<span>…</span>`	Does nothing but provide a place to put `style` or other attributes. (Similar to `<div>…</div>`, but does not cause a line break.)
Attribute	

`style="style"`	Includes inline style specifications. (Can be used in `<span>`, `<div>`, `<body>`, and most other HTML tags.)

Q&A

Q. I've created a web page, but when I open the file in my web browser, I see all the text, including the HTML tags. Sometimes I even see weird gobbledygook characters at the top of the page. What did I do wrong?

A. You didn't save the file as plain text. Try saving the file again, being careful to save it as Text Only or ASCII Text. If you can't quite figure out how to get your word processor to do that, don't stress. Just type your HTML files in Notepad or TextEdit instead, and everything should work just fine. (Also, always make sure that the filename of your web page ends in `.html` or `.htm`.)

Q. I've seen web pages on the Internet that don't have `<!DOCTYPE>` or `<html>` tags at the beginning. You said pages always have to start with these tags. What's the deal?

A. Many web browsers will forgive you if you forget to include the `<!DOCTYPE>` or `<html>` tag and will display the page correctly anyway. However, it's a very good idea to include it because some software does need it to identify the page as valid HTML. Besides, you want your pages to be bona fide HTML pages so that they conform to the latest web standards.

Q. Do I have to use semantic markup at all? Didn't you say throughout this lesson that pages are valid with or without it?

A. True, none of these elements is required for a valid HTML document. You don't have to use any of them, but I urge you to think beyond the use of markup for visual display only and think about it for semantic meaning as well. Visual display is meaningless to screen readers, but

semantic elements convey a ton of information through these machines.

Q. Say I link a style sheet to my page that says all text should be blue, but there's a `<span style="font-color:red">` tag in the page somewhere. Will that text display as blue or red?

A. Red. Local inline styles always take precedence over external style sheets. Any style specifications you put between `<style>` and `</style>` tags at the top of a page also take precedence over external style sheets (but not over inline styles later in the same page). This is the cascading effect of style sheets that I mentioned earlier in the chapter. You can think of cascading style effects as starting with an external style sheet, which is overridden by an internal style sheet, which is overridden by inline styles.

Q. Can I link more than one style sheet to a single page?

A. Sure. For example, you might have a sheet for formatting (text, fonts, colors, and so on) and another one for layout (margins, padding, alignment, and so on). Just be sure to include a `<link>` for both. Technically, the CSS standard requires web browsers to give the user the option to choose between style sheets when multiple sheets are presented via multiple `<link>` tags. However, in practice, all major web browsers simply include every style sheet unless it has a `rel="alternate"` attribute. The preferred technique for linking in multiple style sheets involves using the special `@import` command. The following is an example of importing multiple style sheets with `@import`:

```
@import url(styles1.css);
@import url(styles2.css);
```

Similar to the `<link>` tag, the `@import` command must be placed in the head of a web page.

Workshop

The Workshop contains quiz questions and exercises to help you solidify your understanding of the material covered. Try to answer all questions before looking at the "Answers" section that follows.

Quiz

1. Which five tags does every HTML5 page require?
2. Which of the semantic elements discussed in this chapter is appropriate for containing the definition of a word used in an article?
3. Do you have to use an `<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>`, or `<h6>` element within a `<header>` element?
4. How many different `<nav>` elements can you have in a single page?
5. How many different ways are there to ensure that style rules can be applied to your content?

Answers

1. Every HTML page requires `<html>`, `<head>`, `<title>`, and `<body>` (along with their closing tags, `</html>`, `</head>`, `</title>`, and `</body>`, respectively), plus `<!DOCTYPE html>` on the very first line.
2. The `<aside>` element is appropriate for this situation.
3. No. The `<header>` element can contain any other flow content besides another `<header>` element or a `<footer>` element. However, a heading element (`<h1>` through `<h6>`) is not required in a `<header>` element.
4. You can have as many `<nav>` elements as you need. The trick is to "need" only a few (perhaps for primary and secondary navigation only); otherwise, the meaning is lost.
5. Three: externally, internally, and inline.

Exercises

- Even if your main goal in reading this book is to create web content for your business, you might want to make a personal web page just for practice. Type a few paragraphs to introduce yourself to the world, and use the HTML tags you learned in this chapter to make them into a web page.
- Throughout the book, you'll be following along with the code examples and making pages of your own. Take a moment now to set up a basic document template containing the document type declaration and tags for the core HTML document structure. That way, you can be ready to copy and paste that information whenever you need it.
- Develop a standard style sheet for your website, and link it into all your pages. (Use internal style sheets and/or inline styles for pages that need to deviate from it.) If you work for a corporation, chances are it has already developed font and style specifications for printed materials. Get a copy of those specifications and follow them for company web pages too.