Introduction to JavaScript

In this chapter you will:

- Study the history of the World Wide Web
- Work with well-formed Web pages
- Learn about Web development
- Learn about the JavaScript programming language
- Add structure to your JavaScript programs
- Learn about logic and debugging

The original purpose of the World Wide Web (WWW) was to locate and display information. However, once the Web grew beyond a small academic and scientific community, people began to recognize that greater interactivity would make the Web more useful. As commercial applications of the Web grew, the demand for more interactive and visually appealing Web sites also grew.

To respond to the demand for greater interactivity, an entirely new Web programming language was needed. Netscape filled this need by developing the JavaScript programming language. Originally designed for use in the Navigator Web browser, JavaScript is now also used in most Web browsers including Internet Explorer.

Although JavaScript is considered a programming language, it is also a critical part of Web page design and authoring. This is because the JavaScript language “lives” within a Web page’s elements. In other words, the JavaScript code you write is usually placed within the elements that make up a Web page. JavaScript can turn static documents into applications such as games or calculators. JavaScript code can change the contents of a Web page after a browser has rendered it. It can also create visual effects such as animation, and it can control the Web browser window itself. None of this was possible before the creation of JavaScript.

In this chapter, you will learn the skills required to create basic JavaScript programs. In order to be successful in your JavaScript studies, you should already possess a strong knowledge of HTML and Web page-authoring techniques. The first part of this chapter provides a quick refresher on the history of the World Wide Web and the basics on how to create Web pages with HTML and its successor, XHTML. If you are extremely comfortable with how to create Web pages, then feel free to skip this material. However, be sure that you understand key terms, such as “document” and “element,” because they are used frequently throughout this book.
The World Wide Web

The Internet is a vast network that connects computers all over the world. The original plans for the Internet grew out of a series of memos written by J. C. R. Licklider of Massachusetts Institute of Technology (MIT), in August 1962, discussing his concept of a “Galactic Network.” Licklider envisioned a global computer network through which users could access data and programs from any site on the network. The Internet was actually developed in the 1960s by the Advanced Research Projects Agency (or ARPA) of the U.S. Department of Defense, which later changed its name to Defense Advanced Research Projects Agency (or DARPA). The goal of the early Internet was to connect the main computer systems of various universities and research institutions that were funded by this agency. This first implementation of the Internet was referred to as the ARPANET. More computers were connected to the ARPANET in the years following its initial development in the 1960s, although access to the ARPANET was still restricted by the U.S. government primarily to academic researchers, scientists, and the military.

The 1980s saw the widespread development of local area networks (LANs) and the personal computer. Although at one time restricted to academia and the military, computers and networks soon became common in business and everyday life. By the end of the 1980s, businesses and individual computer users began to recognize the global communications capabilities and potential of the Internet, and convinced the U.S. government to allow commercial access to the Internet.

In 1990 and 1991, Tim Berners-Lee created what would become the World Wide Web, or the Web, at the European Laboratory for Particle Physics (CERN) in Geneva, Switzerland, as a way to easily access cross-referenced documents that existed on the CERN computer network. When other academics and scientists saw the usefulness of being able to easily access cross-referenced documents using Berners-Lee’s system, the Web as we know it today was born. In fact, this method of accessing cross-referenced documents, known as hypertext linking, is probably the most important aspect of the Web because it allows you to open other Web pages quickly. A hypertext link, or hyperlink, contains a reference to a specific Web page that you can click to open that Web page.
A common misconception is that the words “Web” and “Internet” are synonymous. The Web is only one part of the Internet, and is a means of communicating on the Internet. The Internet is also composed of other communication methods such as e-mail systems that send and receive messages. However, due to its enormous influence on computing, communications, and the economy, the World Wide Web is arguably the most important part of the Internet today and is the primary focus of this book.

A document on the Web is called a Web page and is identified by a unique address called the Uniform Resource Locator, or URL. A URL is also commonly referred to as a Web address. A URL is a type of Uniform Resource Identifier (URI), which is a generic term for many types of names and addresses on the World Wide Web. The term Web site refers to the location on the Internet of the Web pages and related files (such as graphic and video files) that belong to a company, organization, or individual. You display a Web page on your computer screen using a program called a Web browser. A person can retrieve and open a Web page in a Web browser either by entering a URL in the Web browser’s Address box or by clicking a hypertext link. When a user wants to access a Web page, either by entering its URL in a browser’s Address box or by clicking a link, the user’s Web browser asks a Web server for the Web page in what is referred to as a request. A Web server is a computer that delivers Web pages. What the Web server returns to the user is called the response.

Web Browsers

You can choose from a number of different browsers, but at the time of this writing, Microsoft Internet Explorer is the most popular browser on the market. Although Internet Explorer is the most popular browser, it was not the first. NCSA Mosaic was created in 1993 at the University of Illinois and was the first program to allow users to navigate the Web using a graphical user interface (GUI). In 1994, Netscape released Navigator, which soon controlled 75% of the market. Netscape maintained its control of the browser market until 1996, when Microsoft entered the market with the release of Internet Explorer, and the so-called browser wars began, in which Microsoft and Netscape fought for control of the browser market.

The browser wars began over DHTML, a combination of various technologies including HTML and JavaScript that allows a Web page to change after it has been rendered by a browser. Examples of DHTML include the ability to position text and elements, change document background color, and create effects such as animation.

Earlier versions of Internet Explorer and Navigator included DHTML elements that were incompatible. Furthermore, Microsoft and Netscape each wanted its version of DHTML to become the industry standard. To settle the argument, the World Wide Web Consortium set out to create a platform-independent and browser-neutral version of DHTML. The World Wide Web
Consortium, or W3C, was established in 1994 at MIT to oversee the development of Web technology standards. While the W3C was drafting a recommendation for DHTML, versions 4 of both Internet Explorer and Navigator added a number of proprietary DHTML elements that were completely incompatible with the other browser. As a result, when working with advanced DHTML techniques such as animation, a programmer had to write a different set of HTML code for each browser type. Unfortunately for Netscape, the W3C adopted as the formal standard the version of DHTML found in version 4 of Internet Explorer, which prompted many loyal Netscape followers to defect to Microsoft.

TIP
DHTML is actually a combination of HTML, Cascading Style Sheets, and JavaScript. The term Cascading Style Sheets (CSS), or style sheets, refers to a standard set by the W3C for managing Web page formatting.

NOTE
The W3C does not actually release a version of a particular technology. Instead, it issues a formal recommendation for a technology, which essentially means that the technology is (or will be) a recognized industry standard.

One great benefit of the browser wars is that it has forced the Web industry to rapidly develop and adopt advanced Web page standards (including JavaScript, CSS, and DHTML) that are consistent across browser types. When the third edition of this book was published in 2004, Internet Explorer appeared to be winning the browser wars as it controlled 95% of the browser market. Yet, in the past few years, Internet Explorer has lost significant market share to a contentious newcomer, Mozilla Firefox. The Firefox Web browser is open source software that is developed by the Mozilla organization (http://www.mozilla.org). Open source refers to software for which the source code can be freely used and modified. At the time of this writing, Internet Explorer usage has slipped to approximately 60%, while Firefox now controls approximately 30% of the market (according to W3 Schools browser statistics page at http://www.w3schools.com/browsers/browsers_stats.asp). One of the most fascinating aspects of Firefox is that it’s essentially an open source version of the Netscape browser. So in a figurative sense, the original Netscape browser has risen from the ashes to resume battle with its arch nemesis, Internet Explorer. Healthy competition is good for any market, so hopefully the renewed hostilities in the browser wars will encourage vendors to continue improving browser quality and capabilities, and to adopt and adhere to Web page standards.

HTML Documents
Originally, people created Web pages using Hypertext Markup Language. Hypertext Markup Language, or HTML, is a markup language used to create the Web pages that appear on the World Wide Web. Web pages are also commonly referred to as HTML pages or documents. A markup language is a set of characters or symbols that define a document’s logical structure—that is, it specifies how a document should be printed or displayed. HTML is based on an older language
called Standard Generalized Markup Language, or SGML, which defines the data in a document independent of how the data will be displayed. In other words, SGML separates the data in a document from the way that data is formatted. Each element in an SGML document is marked according to its type, such as paragraphs, headings, and so on. Like SGML, HTML was originally designed as a way of defining the elements in a document independent of how they would appear. HTML was not intended to be used as a method of designing the actual appearance of the pages in a Web browser. However, HTML gradually evolved into a language that is capable of defining how elements should appear in a Web browser.

NOTE
This textbook uses the terms “Web pages” and “HTML documents” interchangeably.

Basic HTML Syntax
HTML documents are text documents that contain formatting instructions, called tags, which determine how data is displayed on a Web page. HTML tags range from formatting commands that make text appear in boldface or italic, to controls that allow user input, such as option buttons and check boxes. Other HTML tags allow you to display graphic images and other objects in a document or Web page. Tags are enclosed in brackets (< >), and most consist of an opening tag and a closing tag that surround the text or other items they format or control. The closing tag must include a forward slash (/) immediately after the opening bracket to define it as a closing tag. For example, to make a line of text appear in boldface, you use the opening tag <b> and the closing tag </b>. Any text contained between this pair of tags appears in boldface when you open the HTML document in a Web browser.

A tag pair and any data it contains are referred to as an element. The information contained within an element’s opening and closing tags is referred to as its content. Some elements do not require a closing tag. Elements that do not require a closing tag are called empty elements because you cannot use a tag pair to enclose text or other elements. For instance, the <hr> element, which inserts a horizontal rule on a Web page, does not include a closing tag. You simply place the <hr> element anywhere in an HTML document where you want the horizontal rule to appear.

TIP
HTML documents must have a file extension of .html or .htm.
There are literally hundreds of HTML elements. Table 1-1 lists some of the more common elements.

<table>
<thead>
<tr>
<th>HTML element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;b&gt;&lt;/b&gt;</code></td>
<td>Formats enclosed text in a bold typeface</td>
</tr>
<tr>
<td><code>&lt;body&gt;&lt;/body&gt;</code></td>
<td>Encloses the body of the HTML document</td>
</tr>
<tr>
<td><code>&lt;br&gt;</code></td>
<td>Inserts a line break</td>
</tr>
<tr>
<td><code>&lt;center&gt;</code></td>
<td>Centers a paragraph in the middle of a Web page</td>
</tr>
<tr>
<td><code>&lt;head&gt;&lt;/head&gt;</code></td>
<td>Encloses the page header and contains information about the entire page</td>
</tr>
<tr>
<td><code>&lt;hn&gt;&lt;/hn&gt;</code></td>
<td>Indicates heading level elements, where ( n ) represents a number from 1 to 6</td>
</tr>
<tr>
<td><code>&lt;hr&gt;</code></td>
<td>Inserts a horizontal rule</td>
</tr>
<tr>
<td><code>&lt;html&gt;&lt;/html&gt;</code></td>
<td>Begins and ends an HTML document; these are required elements</td>
</tr>
<tr>
<td><code>&lt;i&gt;&lt;/i&gt;</code></td>
<td>Formats enclosed text in an italic typeface</td>
</tr>
<tr>
<td><code>&lt;img src=&quot;...&quot; alt=&quot;...&quot;&gt;</code></td>
<td>Inserts an image file</td>
</tr>
<tr>
<td><code>&lt;p&gt;&lt;/p&gt;</code></td>
<td>Identifies enclosed text as a paragraph</td>
</tr>
<tr>
<td><code>&lt;u&gt;&lt;/u&gt;</code></td>
<td>Formats enclosed text as underlined</td>
</tr>
</tbody>
</table>

Table 1-1  Common HTML elements

All HTML documents must use the `<html>` element as the root element. A root element contains all the other elements in a document. This element tells a Web browser to assemble any instructions between the tags into a Web page. The opening and closing `<html>` tags are required and contain all the text and other elements that make up the HTML document.

Two other important HTML elements are the `<head>` element and the `<body>` element. The `<head>` element contains information that is used by the Web browser, and you place it at the beginning of an HTML document, after the opening `<html>` tag. You place several elements within the `<head>` element to help manage a document’s content, including the `<title>` element, which contains text that appears in a browser’s title bar. A `<head>` element must contain a `<title>` element. With the exception of the `<title>` element, elements contained in the `<head>` element do not affect the display of the HTML document. The `<head>` element and the elements it contains are referred to as the document head.

Following the document head is the `<body>` element. The `<body>` element and the text and elements it contains are referred to as the document body.

When you open an HTML document in a Web browser, the document is assembled and formatted according to the instructions contained in its elements. The process by which a Web browser assembles or formats an HTML document is called parsing or rendering. The following example shows how to make a paragraph appear in boldface in an HTML document:

```html
<p><b>This paragraph will appear in boldface in a Web browser.</b></p>
```
HTML is not case sensitive, so you can use `<B>` in place of `<b>`. However, the next generation of HTML, a language called XHTML, is case sensitive, and you must use lowercase letters for elements. For this reason, this book uses lowercase letters for all elements. (You will learn about XHTML shortly.)

You use various parameters, called attributes, to configure many HTML elements. You place an attribute before the closing bracket of the opening tag, and separate it from the tag name or other attributes with a space. You assign a value to an attribute using the syntax `attribute="value"`. For example, you can configure the `<img>` element, which embeds an image in an HTML document, with a number of attributes, including the `src` attribute. The `src` attribute specifies the filename of an image file or video clip. To include the `src` attribute within the `<img>` element, you type `<img src="mygraphic.gif">`.

When a Web browser parses or renders an HTML document, it ignores nonprinting characters such as tabs and line breaks; the final document that appears in the Web browser includes only recognized HTML elements and text. You cannot use line breaks in the body of an HTML document to insert spaces before and after a paragraph; the browser recognizes only paragraph `<p>` and line break `<br>` elements for this purpose. In addition, most Web browsers ignore multiple, contiguous spaces on a Web page and replace them with a single space. The following code shows a simple HTML document, and Figure 1-1 shows how it appears in a Web browser.

```html
<html>
<head>
<title>Canada</title>
</head>
<body>
<h1>Canada</h1>
<img src="Canada.gif" alt="Image of the Canadian flag.">
<h2>Statistics</h2>
<p><b>Capital</b>: Ottawa<br>
    <b>Largest City</b>: Toronto</p>
<p><b>Official Languages</b>: English, French</p>
<p><b>National Anthem</b>: "O Canada"
    <b>Royal Anthem</b>: "God Save the Queen"
</p>
</body>
</html>
```

The majority of the screen captures of Web pages shown in this book were taken in the Mozilla Firefox Web browser, version 2.0, running on the Windows XP operating system. Different Web browsers may render the parts of a Web page slightly differently from other browsers. The appearance of a Web browser itself can also vary across platforms. If you are using a Web browser other than Firefox and an operating system other than Windows XP, your Web pages and Web browser might not match the figures in this book.
Creating an HTML Document

Because HTML documents are text files, you can create them in any text editor, such as Notepad or WordPad, or any word-processing application capable of creating simple text files. If you use a text editor to create an HTML document, you cannot view the final result until you open the document in a Web browser. Instead of a text editor or word processor, you could choose to use an HTML editor, which is an application designed specifically for creating HTML documents. Some popular HTML editors, such as Macromedia Dreamweaver and Microsoft FrontPage, have graphical interfaces that allow you to create Web pages and immediately view the results, similar to the WYSIWYG (what-you-see-is-what-you-get) interface in word-processing programs. In addition, many current word-processing applications, including Microsoft Word and WordPerfect, allow you to save files as HTML documents.

Like text editors, HTML editors create simple text files, but they automate the process of applying elements. For example, suppose you are creating a document in Word. You can add boldface to a heading in the document simply by clicking a toolbar button. Then, when you save the document as an HTML document, Word automatically adds the `<b>` element to the text in the HTML document.

**TIP**
Many people who are new to creating Web pages are surprised by the fact that you cannot use a Web browser to create an HTML document.
Any HTML editor can greatly simplify the task of creating Web pages. However, HTML editors automatically add many unfamiliar elements and attributes to documents that might confuse you and distract from the learning process. For this reason, in this book you create Web pages using a simple text editor.

Now you are ready to start creating an HTML document that displays the home page for Forestville Funding. The document will contain some of the elements you have seen in this section. You can use any text editor, such as Notepad or WordPad.

Before you begin the first exercise, be certain to extract the data files, located on the CD-ROM that came with this book. Use the 0150-9d.exe file to install the data files on Windows operating systems and the 0150-9d.jar file to install the data files on UNIX/Linux operating systems. You can also download the files from Course Technology’s Web site at http://www.course.com. The 0150-9d.exe and 0150-9d.jar files automatically create directories where you can store the exercises and projects you create in this book and install any necessary data files that you will need. By default, the directories and data files are installed for Windows platforms in C:\Course Technology\0150-9 and for UNIX/Linux platforms in usr/local/course/0150-9. The 0150-9 directory contains separate directories for each chapter, which, in turn, contain the Chapter, Projects, and Cases directories. Figure 1-2 illustrates the Windows directory structure for Chapter 1.

**Figure 1-2**
Windows directory structure for data files

Exercises and projects you create in the main body of each chapter should be saved within the Chapter directory. Save the Hands-on Projects and Case Projects you create at the end of each chapter in the Projects and Cases directories, respectively.

To create a simple HTML document:

1. Start your text editor and create a new document, if necessary.
2. Type the following elements to begin the HTML document. Remember that all HTML documents must begin and end with the `<html>` element.
   ```html
   <html>
   </html>
   ```
3. Next, add the following `<head>` and `<title>` elements between the `<html>...</html>` tag pair. The title appears in your Web browser’s title bar.
Remember that the <head> element must include the <title> element. The <title> element cannot exist outside the <head> element.

```
<head>
<title>Forestville Funding</title>
</head>
```

4. Next, add the following elements above the closing <html> tag. The <body> element contains all of the elements that are rendered in a Web browser.

```
<body>
</body>
```

5. Add the following elements and text above the closing <body> element. The code contains standard HTML elements along with the text that is displayed in the Web browser.

```
<font face="Arial" color="olive">
<h1>Forestville Funding</h1>
<hr>
<h2>Fixed Rates</h2>
<ul>
<li>30-year: 6.25%</li>
<li>15-year: 6.00%</li>
</ul>
</font>
<font face="Arial" color="blue">
<h2>Adjustable Rates</h2>
<ul>
<li>1/1 ARM: 5.26%</li>
<li>3/1 ARM: 5.51%</li>
<li>5/1 ARM: 5.60%</li>
</ul>
</font>
<font face="Arial" color="olive">
<h2>Popular Mortgage Programs</h2>
<ul>
<li>No money down</li>
<li>No income or asset verification</li>
<li>Interest only loans</li>
<li>Home equity loans</li>
</ul>
<hr>
<p>Forestville Funding. Member FDIC. Equal Housing Lender. &copy; 2008 Forestville Funding. All rights reserved.</p>
```

6. Save the document as <ForestvilleFunding.html> in the Chapter folder for Chapter 1. Some text editors automatically add their own extensions to a document. Notepad, for instance, adds an extension of .txt. Be sure your document is saved with an extension of .html. Keep the document open in your text editor.
Some Web servers do not correctly interpret spaces within the name of HTML files. For example, some Web servers may not correctly interpret a filename of Forestville Funding.html, with a space between Forestville and Funding. For this reason, filenames in this book do not include spaces.

7. Open the **ForestvilleFunding.html** document in your Web browser. (You open a local document in most Web browsers by selecting Open or Open File from the File menu.) Figure 1-3 displays the ForestvilleFunding.html document as it appears in Firefox.

8. Close your Web browser window.

**Working with Well-Formed Web Pages**

HTML first became an Internet standard in 1993 with the release of version 1.0. The next version of HTML, 2.0, was released in 1994 and included many core HTML features such as forms and the ability to bold and italicize text. However, many of the standard features that are widely used today, such as using tables to organize text and graphics on a page, were not available until the release of HTML 3.2 in 1996. The current version of HTML, 4.01, was released in 1999. HTML 4.01, however, is the last version of the HTML language; it has been replaced with Extensible Hypertext Markup Language, or XHTML, which is the next generation markup language for creating the Web pages that appear on the World Wide Web.
HTML has been replaced because it is useful only for rendering documents in traditional Web browsers like Firefox or Internet Explorer. That worked well as long as browsers running on computers were the main source of requests for files over the Web. These days, however, many types of devices besides computers use the Web. For example, mobile phones and PDAs are commonly used to browse the Web. An application that is capable of retrieving and processing HTML and XHTML documents is called a user agent. A user agent can be a traditional Web browser or a device such as a mobile phone or PDA, or even an application such as a crawler for a search engine that simply collects and processes data instead of displaying it.

Although user agents other than browsers can process HTML, they are not ideally suited to the task, primarily because HTML is more concerned with how data appears than with the data itself. As Web browsers have evolved over the years, they have added extensions (elements and attributes) to HTML to provide functionality for displaying and formatting Web pages. For instance, one extension to the original HTML language is the `<font>` element, which allows you to specify the font for data in an HTML document. The `<font>` element has nothing to do with the type of data in an HTML document. Instead, its sole purpose is to display data in a specific typeface within a Web browser. There is nothing wrong with continuing to author your Web pages using HTML and design elements such as the `<font>` element—provided your Web pages will be opened only in a Web browser. However, many user agents (such as mobile phones and PDAs) display only black and white or grayscale text and are incapable of processing HTML elements that handle the display and formatting of data. User agents such as these require a language that truly defines data (such as a paragraph or heading) independently of the way it is displayed.

**NOTE**

XHTML is based on Extensible Markup Language, or XML, which is used for creating Web pages and for defining and transmitting data between applications.

The Web page examples and exercises in this book are written in XHTML. Although you need to have a solid understanding of HTML to be successful with this book, you do not necessarily need to be an expert with XHTML. Because XHTML is almost identical to HTML, you can easily adapt any of your existing HTML skills to XHTML, and vice versa.

**TIP**

To ensure backward compatibility with older browsers, you should save XHTML documents with an extension of `.html` or `.htm`, just like HTML documents.

**XHTML Document Type Definitions (DTDs)**

When a document conforms to the rules and requirements of XHTML, it is said to be well formed. Among other things, a well-formed document must include a `<!DOCTYPE>` declaration and the `<html>`, `<head>`, and `<body>` elements. The `<!DOCTYPE>` declaration belongs in the first line of an XHTML document and determines the Document Type Definition with which the
document complies. A Document Type Definition, or DTD, defines the elements and attributes that can be used in a document, along with the rules that a document must follow when it includes them. You can use three types of DTDs with XHTML documents: transitional, strict, and frameset.

To understand the differences among the three types of DTDs, you need to understand the concept of deprecated HTML elements. One of the goals of XHTML is to separate the way HTML is structured from the way the parsed Web page is displayed in the browser. To accomplish this goal, the W3C decided that several commonly used HTML elements and attributes for display and formatting would not be used in XHTML 1.0. Instead of using HTML elements and attributes for displaying and formatting Web pages, the W3C recommends you use Cascading Style Sheets (CSS), which are discussed later in this chapter.

Elements and attributes that are considered obsolete and that will eventually be eliminated are said to be deprecated. Table 1-2 lists the HTML elements that are deprecated in XHTML 1.0.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;applet&gt;</td>
<td>Executes Java applets</td>
</tr>
<tr>
<td>&lt;basefont&gt;</td>
<td>Specifies the base font size</td>
</tr>
<tr>
<td>&lt;center&gt;</td>
<td>Centers text</td>
</tr>
<tr>
<td>&lt;dir&gt;</td>
<td>Defines a directory list</td>
</tr>
<tr>
<td>&lt;font&gt;</td>
<td>Specifies a font name, size, and color</td>
</tr>
<tr>
<td>&lt;isindex&gt;</td>
<td>Creates automatic document indexing forms</td>
</tr>
<tr>
<td>&lt;menu&gt;</td>
<td>Defines a menu list</td>
</tr>
<tr>
<td>&lt;s&gt; or &lt;strike&gt;</td>
<td>Formats strikethrough text</td>
</tr>
<tr>
<td>&lt;u&gt;</td>
<td>Formats underlined text</td>
</tr>
</tbody>
</table>

Table 1-2  HTML elements that are deprecated in XHTML 1.0

The three DTDs are distinguished in part by the degree to which they accept or do not accept deprecated HTML elements. This is explained in more detail in the following sections.

**Transitional DTD**

The transitional DTD allows you to use deprecated style elements in your XHTML documents. The <!DOCTYPE> declaration for the transitional DTD is as follows:

```html
<!DOCTYPE html PUBLIC
"-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

You should use the transitional DTD only if you need to create Web pages that use the deprecated elements listed in Table 1-2.

**Frameset DTD**

The frameset DTD is identical to the transitional DTD, except that it includes the <frameset> and <frame> elements, which allow you to split the browser window into two or more frames. The <!DOCTYPE> declaration for the frameset DTD is as follows:
You should understand that frames have been deprecated in favor of tables. However, frameset documents are still widely used, and you need to be able to recognize and work with them in the event that you need to modify an existing Web page that was created with frames.

**Strict DTD**
The strict DTD eliminates the elements that were deprecated in the transitional DTD and frameset DTD. The `<!DOCTYPE>` declaration for the strict DTD is as follows:

```xml
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

As a rule, you should always try to use the strict DTD. This ensures that your Web pages conform to the most current Web page authoring techniques. Next, you add a `<!DOCTYPE>` declaration for the strict DTD to the Forestville Funding page.

To add a `<!DOCTYPE>` declaration for the strict DTD to the Forestville Funding page:

1. Return to the `ForestvilleFunding.html` document in your text editor.
2. Add the following `<!DOCTYPE>` declaration for the strict DTD as the first line in the document (above the opening `<html>` tag):

   ```xml
   <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
   ```

3. Save your changes to the document.

**Writing Well-Formed Documents**
As you learned earlier, a well-formed document must include a `<!DOCTYPE>` declaration and the `<html>`, `<head>`, and `<body>` elements. The following list describes some other important components of a well-formed document:

- All XHTML documents must use `<html>` as the root element. The `xmlns` attribute is required in the `<html>` element and must be assigned the `http://www.w3.org/1999/xhtml` URI.
- XHTML is case sensitive.
- All XHTML elements must have a closing tag.
- Attribute values must appear within quotation marks.
- Empty elements must be closed.
- XHTML elements must be properly nested.
Most of the preceding rules are self-explanatory. However, the last rule requires further explanation. **Nesting** refers to how elements are placed inside other elements. For example, in the following code, the `<i>` element is nested within the `<b>` element, while the `<b>` element is nested within a `<p>` element.

```html
<p><b><i>Call for a free estimate!</i></b></p>
```

In an HTML document, it makes no difference how the elements are nested. Examine the following modified version of the preceding statement:

```html
<p><b><i>Call for a free estimate!</i></b></p></i>
```

In this version, the opening `<i>` element is nested within the `<b>` element, which, in turn, is nested within the `<p>` element. Notice, however, that the closing `</i>` tag is outside the closing `</p>` tag. The `<i>` is the innermost element. In XHTML, the innermost element in a statement must be closed before another element is closed. In the preceding statement, the `<b>` and `<p>` elements are closed before the `<i>` element. Although the order in which elements are closed makes no difference in HTML, the preceding code would prevent an XHTML document from being well formed.

The second-to-last rule in the list (“Empty elements must be closed.”) also requires further explanation. Three of the most common empty elements in HTML are the `<hr>` element, which inserts a horizontal rule into the document, the `<br>` element, which inserts a line break, and the `<img>` element, which adds an image to the document. You close an empty element in XHTML by adding a space and a slash before the element’s closing bracket. For example, the following code shows how to use the `<hr>` and `<br>` elements in an XHTML document. Figure 1-4 shows how the code appears in a Web browser.

```html
<hr />
<p>In 2005, <b>ESPN</b> had 89.9 million subscribers, <br />
<b>A&E Network</b> had 89.0 million subscribers, <br />
and <b>The History Channel</b> had 88.2 million subscribers.</p>
<hr />
```

**Figure 1-4**
XHTML document with closed empty elements

NOTE
You might be wondering why XHTML documents do not use a root element of `<xhtml>`. The `<html>` element is necessary for backward compatibility with older browsers that do not recognize the `<!DOCTYPE>` element, which declares the DTD used by an XHTML element.
Next, you modify the Forestville Funding page so it is well formed. As you complete the following steps, note that, in this book, boldface is used to indicate code that you need to insert within other code. For example, in Step 2 in the following steps, you should type the boldface code after <html in the <html> element.

To modify the Forestville Funding page so it is well formed:

1. Return to the ForestvilleFunding.html document in your text editor.
2. Modify the opening <html> element so it includes the xmlns attribute. Insert the boldface code into the <html> element as follows:
   
   ```html
   <html xmlns="http://www.w3.org/1999/xhtml">
   ```
   
3. Delete the six <font> elements. Be certain to also delete the closing </font> tags. (You need to do this because the <font> element has been deprecated in XHTML in favor of CSS.)
4. Add a space and a slash before the closing bracket of the two <hr> elements. Also add a space and a slash before the closing bracket of the <br> element in the last paragraph.
5. Save the ForestvilleFunding.html document and open it in your Web browser. The document should appear similar to the way it did before you made it well formed, although it will not contain any formatting. You learn how to add formatting when you study CSS in the next section.

Using Phrase Elements

Recall that early on, Web browser makers began to add their own extensions to HTML in order to provide functionality for displaying and formatting Web pages. These extensions (such as the bold and font elements) did nothing to describe the type of data being presented, but only served to instruct a Web browser how to display and format it. At the time, these extensions were considered a useful improvement. But as user agents become more complex, more nuanced elements became necessary. For example, consider the bold element. Visually, it’s a great way to emphasize a word or phrase. However, it’s not so useful for a user agent for the visually impaired that reads the contents of a Web page out loud. The Web developer needs some way of telling this type of user agent which text should receive extra, audible emphasis.

To address this type of issue, XHTML uses two types of inline elements for managing the formatting of text in an XHTML document: formatting elements and phrase elements. Formatting elements provide specific instructions about how their contents should be displayed. Two of the most commonly used formatting elements are the <b> element (for boldface) and the <i> element (for italics). Phrase elements, on the other hand, primarily identify or describe their contents. For instance, the <em> element is an emphasized piece of data, similar to a quotation. How the <em> element is rendered is up to each user agent, although most current Web browsers display the contents of the <em> element using italics. However, a user agent for the vision impaired may use the <em> element to pronounce the text or phrase it contains with more emphasis, in order to get the meaning across to the vision-impaired visitor to the Web site. Although text-formatting elements are commonly used and work perfectly well for displaying text with a specific style of formatting, it’s better to format the text on your Web pages using a phrase element that describes
its content. Using phrase elements helps ensure that your Web pages are compatible with user agents that may not be capable of handling formatting elements. Generally, you should strive not to use formatting elements at all and use only CSS to manage the display of elements on your Web pages. However, because several of the basic formatting elements are so commonly used, they are not deprecated in XHTML Strict.

Table 1-3 lists the phrase elements that are available in XHTML, along with how each element is rendered by most Web browsers.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
<th>Renders as</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;abbr&gt;</td>
<td>Specifies abbreviated text</td>
<td>Default text</td>
</tr>
<tr>
<td>&lt;acronym&gt;</td>
<td>Identifies an acronym</td>
<td>Default text</td>
</tr>
<tr>
<td>&lt;cite&gt;</td>
<td>Defines a citation</td>
<td>Italic</td>
</tr>
<tr>
<td>&lt;code&gt;</td>
<td>Identifies computer code</td>
<td>Monospace</td>
</tr>
<tr>
<td>&lt;dfn&gt;</td>
<td>Marks a definition</td>
<td>Italic</td>
</tr>
<tr>
<td>&lt;em&gt;</td>
<td>Defines emphasized text</td>
<td>Italic</td>
</tr>
<tr>
<td>&lt;kbd&gt;</td>
<td>Indicates text that is to be entered by a visitor to a Web site</td>
<td>Monospace</td>
</tr>
<tr>
<td>&lt;q&gt;</td>
<td>Defines a quotation</td>
<td>Italic</td>
</tr>
<tr>
<td>&lt;samp&gt;</td>
<td>Identifies sample computer code</td>
<td>Monospace</td>
</tr>
<tr>
<td>&lt;strong&gt;</td>
<td>Defines strongly emphasized text</td>
<td>Bold</td>
</tr>
<tr>
<td>&lt;var&gt;</td>
<td>Defines a variable</td>
<td>Italic</td>
</tr>
</tbody>
</table>

Table 1-3 Phrase elements

Cascading Style Sheets (CSS)

Although you should always strive to create Web pages that are compatible with all user agents, you can also design and format them so they are visually pleasing when rendered in a traditional Web browser. To design and format Web pages for traditional Web browsers, you use CSS, a standard set by the W3C for managing the design and formatting of Web pages in a Web browser. A single piece of CSS formatting information, such as text alignment or font size, is referred to as a style. Some of the style capabilities of CSS include the ability to change fonts, backgrounds, and colors, and to modify the layout of elements as they appear in a Web browser.

CSS information can be added directly to documents or stored in separate documents and shared among multiple Web pages. The term “cascading” refers to the ability of Web pages to use CSS information from more than one source. When a Web page has access to multiple CSS sources, the styles “cascade,” or “fall together.” Keep in mind that CSS design and formatting techniques are truly independent of the content of a Web page, unlike text-formatting elements, such as the <b> and <i> elements. CSS allows you to provide design and formatting specifications for well-formed documents that are compatible with all user agents.

**NOTE**

Entire books are devoted to CSS. This chapter provides only enough information to get you started. To learn more about CSS techniques, refer to Don Gosselin’s *XHTML*, also published by Course Technology. For other books that cover CSS more fully, search for “css” on the Course Technology Web site at [http://www.course.com](http://www.course.com). You can also find the latest information on CSS at the W3C’s Web site: [http://www.w3.org/style/css/](http://www.w3.org/style/css/).
CSS Properties

CSS styles are created with two parts separated by a colon: the property, which refers to a specific CSS style, and the value assigned to it, which determines the style’s visual characteristics. Together, a CSS property and the value assigned to it are referred to as a declaration or style declaration. Figure 1-5 shows a simple style declaration for the color property that changes the color of an element’s text to blue.

![Figure 1-5](image)

**Figure 1-5**
Style declaration

```
property: value
```

Inline Styles

When you design a Web page, you often want the elements on your page to share the same formatting. For example, you might want all of the headings to be formatted in a specific font and color. Later in this section, you will learn how to use internal and external style sheets to apply the same formatting to multiple elements on a Web page. However, there might be times when you want to change the style of a single element on a Web page. The most basic method of applying styles is to use inline styles, which allow you to add style information to a single element in a document. You use the `style` attribute to assign inline style information to an element. You assign to the `style` attribute a property declaration enclosed in quotation marks.

Suppose you want to modify a single paragraph in a document so it uses the Verdana font instead of the browser’s default font. You can modify the default font using the following statement, which uses an inline style declaration for the `font-family` property. Figure 1-6 shows how the paragraph appears in a Web browser.

```
<p>This paragraph does not use CSS.</p>
<p style="font-family: Verdana">Paragraph formatted with inline styles.</p>
```

![Figure 1-6](image)

**Figure 1-6**
Paragraph formatted with an inline style declaration

**NOTE**
The styles you assign to an element are automatically passed to any nested elements it contains. For example, if you use the `font-family` style to assign a font to a paragraph, that font is automatically assigned to any nested elements the paragraph contains, such as `<strong>` or `<em>` elements.
You can include multiple style declarations in an inline style by separating each declaration with a semicolon. The following statement shows the same paragraph element shown earlier, but this time with two additional style declarations: one for the color property, which sets an element’s text color to blue, and one for the text-align property, which centers the paragraph in the middle of the page. Notice that the <strong> element, which is nested in the paragraph element, automatically takes on the paragraph element’s style elements. Figure 1-7 shows how the paragraph appears in a Web browser.

```
<p style="font-family: Verdana; color: blue; text-align: center">Paragraph formatted with <strong>inline styles</strong></p>
```

![Paragraph formatted with multiple inline style declarations](image_url)

Next, you modify the Forestville Funding page so it includes inline styles. You add some simple CSS formatting instructions that format the Web page in the Arial font, the headings in the color olive, and the body text in the color blue.

To modify the Forestville Funding page so it includes inline styles:

1. Return to the `ForestvilleFunding.html` document in your text editor.
2. Modify the opening `<body>` tag, so it includes inline styles that modify the font-family, color, and background-color properties, as follows:

   ```html
   <body style="font-family: Arial; color: blue; background-color: transparent">
   ```

3. Modify the opening `<h1>` tag, so it includes inline styles that modify the font-family and color properties, as follows:

   ```html
   <h1 style="font-family: Arial; color: olive">
   ```

4. Finally, modify the three opening `<h2>` tags, so they include the same styles as the `<h1>` tag. Each opening `<h2>` tag should appear as follows:

   ```html
   <h2 style="font-family: Arial; color: olive">
   ```

5. Save the `ForestvilleFunding.html` document and open it in your Web browser. The Web page should appear the same as the HTML version you created with deprecated formatting elements, but this time the font face appears in Arial.


One of the great advantages to using CSS is that you can share styles among multiple Web pages, making it easier to create and maintain a common look and feel for an entire Web site. Inline styles, however, cannot be shared by other Web pages or even by other elements on the same page.
(except by elements that are nested within other elements). Plus, it is extremely time consuming to add inline styles to each and every element on a Web page. Inline styles are only useful if you need to make a one-time change to a single element on a page. If you want to apply the same formatting to multiple elements on a page or share styles with other Web pages, then you need to use internal or external style sheets.

**Internal Style Sheets**
You use an internal style sheet to create styles that apply to an entire document. You create an internal style sheet within a `<style>` element placed within the document head. The `<style>` element must include a `type` attribute, which is assigned a value of "text/css", as follows:

```
<style type="text/css">
style declarations
</style>
```

Within the `<style>` element, you create any style instructions for a specific element that are applied to all instances of that element contained in the body of the document. The element to which specific style rules in a style sheet apply is called a selector. You create a style declaration for a selector in an internal style sheet by placing a list of declarations within a pair of braces `{ }` following the name of the selector. Figure 1-8 shows some style declarations for the `<p>` element (which is the selector), which change the `color` property to blue.

**Figure 1-8**
Selector style declaration

```
p { color: blue }
```

As with inline styles, you separate multiple properties for a selector by semicolons. The following code shows a portion of the Canada Statistics page you saw earlier, but this time it includes an internal style sheet for the `h1`, `h2`, and `p` selectors. A pair of braces containing style instructions follows each selector. All instances of the associated elements in the body of the document are formatted using these style instructions. Figure 1-9 shows how the document appears in a Web browser.

```
<html>
<head>
<title>Canada</title>
```
You can also group selectors so they share the same style declarations by separating each selector with a comma. For example, you use the following single declaration to format all of a document’s <h1>, <h2>, and <h3> elements to use the same color:

```html
<style type="text/css">
h1, h2, h3 { color: navy }
</style>
```

Next, you modify the Forestville Funding page so it contains an internal style sheet.

To modify the the Forestville Funding page so it contains an internal style sheet:

1. Return to the `ForestvilleFunding.html` document in your text editor.
2. Delete the inline styles in the `<body>`, `<h1>`, and `<h2>` tags.
3. Add the following internal style sheet above the closing `</head>` tag:

```html
<style type="text/css">
body { font-family: Arial; color: blue; background-color: transparent }
h1, h2 { font-family: Arial; color: olive }
</style>
```
4. Save the **ForestvilleFunding.html** document and open it in your Web browser. The Web page should format the same as it did with the inline styles.

5. Close your Web browser window.

**External Style Sheets**

Inline styles are useful if you need to format only a single element; internal style sheets are useful for creating styles that apply to an entire document. However, most companies want all of the documents on a Web site to have the same look and feel. For this reason, it’s preferable to use **external style sheets**, which are separate text documents containing style declarations that are used by multiple documents on a Web site. You should create an external style sheet whenever you need to use the same styles on multiple Web pages in the same site.

You create an external style sheet in a text editor, the same as when you create XHTML documents. However, you should save the document with an extension of .css. The style sheet document should not contain XHTML elements, only style declarations. Use the same rules for creating style declarations in an external style sheet as you use in an internal style sheet. The contents of a typical external style sheet may appear as follows. Notice that the code contains no XHTML elements.

```html
h1 {color: navy; font-size: 2em; font-family: serif}
h2 {color: red; font-size: 1.5em; font-family: Arial}
body {color: blue; font-family: Arial;
  font-size: .8em; font-weight: normal}
```

The most popular way to access the styles in an external style sheet is to use the empty `<link>` element to link a document to a style sheet. You place the `<link>` element in the document head. You include three attributes in the `<link>` element: an `href` attribute that is assigned the URL of the style sheet, the `rel` attribute that is assigned a value of "stylesheet" to specify that the referenced file is a style sheet, and the `type` attribute, which is assigned the same "text/css" value as the `type` attribute used in the `<style>` element. For example, to link a document to a style sheet named `company_branding.css`, you include a link element in the document head, as follows:

```html
<head>
  ...
  <link rel="stylesheet" href="company_branding.css" type="text/css" />
</head>
```

Next, you modify the Forestville Funding page so it is formatted with an external style sheet.

**To modify the Forestville Funding page so it is formatted with an external style sheet:**

1. Return to the **ForestvilleFunding.html** document in your text editor.
2. Copy the style declarations within the `<style>` element and create a new document in your text editor. Be certain not to copy the `<style>` tags.
3. Paste the contents into the new file.
4. Save the file as `forestvillefunding_styles.css` in your Chapter folder for Chapter 1.
5. Close the `forestvillefunding_styles.css` file and return to the `ForestvilleFunding.html` file in your text editor.
6. Replace the `<style>` element and the style declarations it contains with the following `<link>` element that links to the `forestvillefunding_styles.css` external style sheet:

```html
<link rel="stylesheet" href="forestvillefunding_styles.css" type="text/css" />
```
7. Save the `ForestvilleFunding.html` document and open it in your Web browser. The file should appear the same as it did before you linked it to the external style sheet.
8. Close your Web browser window.

Next, you will learn about `content-type` `<meta>` elements, which the W3C strongly encourages you to use to specify an XHTML document’s character set.

**The Content-Type `<meta>` Element**

When a user enters a URL for a Web page in a browser’s Address box or clicks a link to a Web page, the user’s Web browser asks the Web server for the Web page. One part of the response from the Web server is the requested Web page. Another important part of the response is the response header, which is sent to the Web browser before the Web page is sent to provide information that the browser needs to render the page. One of the most important pieces of information in the response header is the type of data, or content type, that the server is sending. For Web pages, you create a content-type `<meta>` element to specify a content type that the document uses. The term metadata means information about information. In a Web page, you use the `<meta>` element to provide information about the information in a Web page. You must place the `<meta>` element within the `<head>` element. You can use three primary attributes with the `<meta>` element: `name`, `content`, and `http-equiv`.

Another important use of the content-type `<meta>` element is to specify a document’s character encoding. This allows a Web server to construct a response header in the appropriate character set. To create a content-type `<meta>` element, you assign a value of `content-type` to the `http-equiv` attribute in a `<meta>` element. You then assign to the `<meta>` element’s `content` attribute a value of `text/html; charset=iso-8859-1`. This specifies that the document’s MIME type is “text/html” and that the document uses the iso-8859-1 character set, which represents English and many western European languages. The following statement shows how to construct the content-type `<meta>` elements:

```html
<meta http-equiv="content-type"
     content="text/html; charset=iso-8859-1" />
```
NOTE
MIME is a protocol that was originally developed to allow different file types to be transmitted as attachments to e-mail messages. Now, MIME has become a standard method of exchanging files over the Internet. You specify MIME types with two-part codes separated by a forward slash (/). The first part specifies the MIME type, and the second part specifies the MIME subtype.

The W3C strongly encourages the use of content-type <meta> elements to specify an XHTML document’s character set. However, a content-type <meta> element is not required because most current Web browsers can determine on their own the character set of an XHTML document. For XHTML documents you create in this book, you include the content-type <meta> element in order to comply with the W3C’s recommendation.

The content-type <meta> element is just one of many response headers that you can construct with the http-equiv attribute. Go to http://vancouver-webpages.com/META/ for a complete list of other response header <meta> elements to use with the http-equiv attribute.

Next, you add the content-type <meta> element to the Forestville Funding page.

To add the content-type <meta> element:

1. Return to the ForestvilleFunding.html document in your text editor.
2. Add the following content type <meta> element above the closing </head> tag:
   `<meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />

3. Save the ForestvilleFunding.html document.

Validating Web Pages
When you open an XHTML document that is not well formed in a Web browser, the browser simply ignores the errors, as it would with an HTML document with errors, and renders the Web page as best it can. The Web browser cannot tell whether the XHTML document is well formed. To ensure that a Web page is well formed and that its elements are valid, you need to use a validating parser. A validating parser is a program that checks whether a Web page is well formed and whether the document conforms to a specific DTD. The term validation refers to the process of verifying that your document is well formed and checking that the elements in your document are correctly written according to the element definitions in a specific DTD. If you do not validate a document and it contains errors, most Web browsers will probably treat it as an HTML document, ignore the errors, and render the page anyway. However, validation can help you spot errors in your code. Even the most experienced Web page authors frequently introduce typos or some other error into a document that prevent the document from being well formed.

Various Web development tools, including Macromedia Dreamweaver, offer validation capabilities. In addition, several validating services can be found online. One of the best available is W3C
Markup Validation Service, a free service that validates both HTML and XHTML. The W3C Markup Validation Service is located at http://validator.w3.org/. The main Web page for the service, shown in Figure 1-10, allows you to validate a Web page by entering its URL or by uploading a document from your computer.

Next, you validate the Forestville Funding page using the W3C Markup Validation Service.

To validate the Forestville Funding page using the W3C Markup Validation Service:

1. Start your Web browser, and enter the Web address for the W3C Markup Validation Service: http://validator.w3.org/.

2. In the Validate by File Upload section, click the Browse button to display the File Upload dialog box. In the File Upload dialog box, locate the ForestvilleFunding.html document in the Chapter folder for Chapter 1. Once you locate the document, double-click it or click it once and select the Open button. The drive, folder path, and file name should appear in the File text box on the upload page. Click the Check button. The W3C Markup Validation Service validates the document and returns the results displayed in Figure 1-11. If you receive any errors, fix them, resave the document, and then revalidate the page.

3. Close your Web browser window.
Understanding Web Development

Web page design, or Web design, refers to the visual design and creation of the documents that appear on the World Wide Web. Most businesses today—both prominent and small—have Web sites. To attract and retain visitors, and to stand out from the crowd, Web sites must be exciting and visually stimulating. Quality Web design plays an important role in attracting first-time and repeat visitors. However, the visual aspect of a Web site is only one part of the story. Equally important is the content of the Web site and how that content is structured.

Web design is an extremely important topic. However, this book is not about Web design, even though you will certainly learn many Web design concepts and techniques as you work through the chapters ahead. Instead, this book touches on both Web page authoring and Web development.

Web page authoring (or Web authoring) refers to the creation and assembly of the tags, attributes, and data that make up a Web page. There is a subtle, but important distinction between Web design and Web page authoring: Web design refers to the visual and graphical design aspects of creating Web pages, whereas a book on Web page authoring refers to the physical task of assembling the Web page tags and attributes. Web development, or Web programming, refers to the design of software applications for a Web site. Generally, a Web developer works “behind the scenes” to develop software applications that access databases and file systems, communicate with other applications, and perform other advanced tasks. The programs created by a Web developer will not necessarily be seen by a visitor to a Web site, although the visitor will certainly use a Web developer’s programs, particularly if the Web site writes and reads data to and from a database. Although JavaScript lives more in the realm of Web page authoring, there is certainly
some overlap between Web authoring and Web development, especially when it comes to sending and receiving data to and from a Web server.

**NOTE**

Another term that you might often see in relation to Web development is “Webmaster.” Although there is some dispute over exactly what the term means, typically Webmaster refers to a person who is responsible for the day-to-day maintenance of a Web site including the monitoring of Web site traffic and ensuring that the Web site’s hardware and software are running properly. The duties of a Webmaster often require knowledge of Web page design, authoring, and development.

**TIP**

If you would like to study the topic of Web page design itself, refer to Joel Sklar’s excellent book, *Web Design*, published by Course Technology.

There are countless ways of combining the hundreds of HTML tags to create interesting Web pages. One technique that professional Web authors use to increase their HTML skill is examining the underlying HTML tags of a Web page that they admire. All Web browsers contain commands that allow you to view the underlying HTML code for a Web page that appears in the browser; in Firefox you select Page Source from the View menu and in Internet Explorer you select the Source command from the View menu.

The open nature of HTML makes it possible for anyone to easily see how another Web author created a Web page. However, you should never copy another Web page author’s work and attempt to pass it off as your own. As a responsible member of the Web community, you should examine the HTML code behind a Web page only to improve your own skills. The potential theft of another Web page author’s hard work and intellectual property is no small concern. Not only is stealing another Web page author’s code and Web page designs unscrupulous, but in many cases it is illegal, especially if the work is copyrighted. Throughout this book you will examine the underlying HTML code from various published Web sites. However, remember that your reasons for examining existing HTML code should be to understand the techniques used to create specific elements on a Web page in order to improve your own skills, not to hijack someone else’s hard work.

**Client/Server Architecture**

To be successful in Web development, you need to understand the basics of client/server architecture. There are many definitions of the terms “client” and “server.” In traditional client/server architecture, the server is usually some sort of database from which a client requests information. A server fulfills a request for information by managing the request or serving the requested information to the client—hence the term, client/server. A system consisting of a client and a server is known as a two-tier system.
One of the primary roles of the client, or front end, in a two-tier system is the presentation of an interface to the user. The user interface gathers information from the user, submits it to a server, or back end, then receives, formats, and presents the results returned from the server. The main responsibility of a server is usually data storage and management. On client/server systems, heavy processing, such as calculations, usually takes place on the server. As desktop computers become increasingly powerful, however, many client/server systems have begun placing at least some of the processing responsibilities on the client. In a typical client/server system, a client computer may contain a front end that is used for requesting information from a database on a server. The server locates records that meet the client request, performs some sort of processing, such as calculations on the data, and then returns the information to the client. The client computer can also perform some processing, such as building the queries that are sent to the server or formatting and presenting the returned data. Figure 1-12 illustrates the design of a two-tier client/server system.

**Figure 1-12**
The design of a two-tier client/server system

The Web is built on a two-tier client/server system, in which a Web browser (the client) requests documents from a Web server. The Web browser is the client user interface. You can think of the Web server as a repository for Web pages. After a Web server returns the requested document, the Web browser (as the client user interface) is responsible for formatting and presenting the document to the user. The requests and responses through which a Web browser and Web server communicate happen with HTTP. For example, if a Web browser requests the URL http://www.course.com, the request is made with HTTP because the URL includes the HTTP protocol. The Web server then returns to the Web browser an HTTP response containing the response header and the HTML (or XHTML) for Course Technology's home page.

After you start adding databases and other types of applications to a Web server, the client/server system evolves into what is known as a three-tier client architecture. A three-tier, or multitier, client/server system consists of three distinct pieces: the client tier, the processing tier, and the data storage tier. The client tier, or user interface tier, is still the Web browser. However, the database portion of the two-tier client/server system is split into a processing tier and the data storage tier. The processing tier, or middle tier, handles the interaction between the Web browser client and the data storage tier. (The processing tier is also sometimes called the processing bridge.) Essentially, the client tier makes a request of a database on a Web server. The processing tier performs any necessary processing or calculations based on the request from the client tier, and then reads information from or writes information to the data storage tier. The processing tier also handles the return of any information to the client tier. Note that the processing tier is not the only place where processing can occur. The Web browser (client tier) still renders Web page documents (which requires processing), and the database or application in the data storage tier
might also perform some processing. Figure 1-13 illustrates the design of a three-tier client/server system.

Figure 1-13
The design of a three-tier client/server system

Client tier
- Handles user interface display (the Web browser) and submits requests to the processing tier

Processing tier
- Handles interaction between the Web browser client and the data storage tier

Data storage tier
- Stores data in a database and returns requests presented by the processing tier

Can be the same computer

TIP
Two-tier client/server architecture is a physical arrangement in which the client and server are two separate computers. Three-tier client/server architecture is more conceptual than physical, because the storage tier can be located on the same server.

NOTE
Multitier client/server architecture is also referred to as n-tier architecture.

JavaScript and Client-Side Scripting

As mentioned earlier, HTML was not originally intended to control the appearance of pages in a Web browser. When HTML was first developed, Web pages were static—that is, they couldn’t change after the browser rendered them. However, after the Web grew beyond a small academic and scientific community, people began to recognize that greater interactivity and better visual design would make the Web more useful. As commercial applications of the Web grew, the demand for more interactive and visually appealing Web sites also grew.

HTML and XHTML could only be used to produce static documents. You can think of a static Web page written in HTML or XHTML as being approximately equivalent to a document created in a word-processing or desktop publishing program; the only thing you can do with it is view it or print it. Thus, to respond to the demand for greater interactivity, an entirely new Web programming language was needed. Netscape filled this need by developing JavaScript.

JavaScript is a client-side scripting language that allows Web page authors to develop interactive Web pages and sites. Client-side scripting refers to a scripting language that runs on a local
browser (on the client tier) instead of on a Web server (on the processing tier). Originally designed for use in Navigator Web browsers, JavaScript is now also used in most other Web browsers, including Firefox and Internet Explorer.

The term scripting language is a general term that originally referred to fairly simple programming languages that did not contain the advanced programming capabilities of languages such as Java or C++. When it comes to Web development, the term scripting language refers to any type of language that is capable of programmatically controlling a Web page or returning some sort of response to a Web browser. It’s important to note that although the term scripting language originally referred to simple programming languages, today’s Web-based scripting languages are anything but simple. The part of a browser that executes scripting language code is called the browser’s scripting engine. A scripting engine is just one kind of interpreter, with the term interpreter referring generally to any program that executes scripting language code. When a scripting engine loads a Web page, it interprets any programs written in scripting languages, such as JavaScript. A Web browser that contains a scripting engine is called a scripting host. Firefox and Internet Explorer are examples of scripting hosts that can run JavaScript programs.

JavaScript was first introduced in Navigator and was originally called LiveScript. With the release of Navigator 2.0, the name was changed to JavaScript 1.0. Subsequently, Microsoft released its own version of JavaScript in Internet Explorer 4.0 and named it JScript.

When Microsoft released JScript, several major problems occurred. For example, the Netscape and Microsoft versions of the JavaScript language differed so greatly that programmers were required to write almost completely different JavaScript programs for Navigator and Internet Explorer. To avoid similar problems in the future, an international, standardized version of JavaScript, called ECMAScript, was created. The most recent version of ECMAScript is edition 3. Both Netscape JavaScript and Microsoft JScript conform to ECMAScript edition 3. Nevertheless, Netscape JavaScript and Microsoft JScript each include unique programming features that are not supported by the other language. In this book, you will learn to create JavaScript programs with ECMAScript edition 3, which is supported by all current Web browsers including Firefox, Netscape 6 and higher, and Internet Explorer 4 and higher.

**TIP**
The next major edition of the JavaScript language is ECMAScript Edition 4. At the time of this writing, the developers of the language have not made a great deal of progress on the new version and it is not known when it will be complete.

**NOTE**
Many people think that JavaScript is related to or is a simplified version of the Java programming language. However, the languages are entirely different. Java is an advanced programming language that was created by Sun Microsystems and is considerably more difficult to master than JavaScript. Although Java can be used to create programs that can run from a Web page, Java programs are usually external programs that execute independently of a browser. In contrast, JavaScript programs always run within a Web page and control the browser.
Although JavaScript is considered a programming language, it is also a critical part of Web page authoring. This is because the JavaScript language “lives” within a Web page's elements. JavaScript gives you the ability to:

- Turn static Web pages into applications such as games or calculators.
- Change the contents of a Web page after a browser has rendered it.
- Create visual effects such as animation.
- Control the Web browser window itself.

For security reasons, the JavaScript programming language cannot be used outside of the Web browser. For example, to prevent mischievous scripts from stealing information, such as your e-mail address or credit card information you use for an online transaction, or from causing damage by changing or deleting files, JavaScript does not allow any file manipulation whatsoever. Similarly, JavaScript does not include any sort of mechanism for creating a network connection or accessing a database. This limitation prevents JavaScript programs from infiltrating a private network or intranet from which information might be stolen or damaged. Another helpful limitation is the fact that JavaScript cannot run system commands or execute programs on a client. The ability to read and write cookies is the only type of access to a client that JavaScript has. Web browsers, however, strictly govern cookies and do not allow access to cookies from outside the domain that created them. This security also means that you cannot use JavaScript to interact directly with Web servers that operate at the processing tier. Although the programmer can employ a few tricks (such as forms and query strings) to allow JavaScript to interact indirectly with a Web server, if you want true control over what’s happening on the server, you need to use a server-side scripting language.

**Server-Side Scripting and PHP**

Server-side scripting refers to a scripting language that is executed from a Web server. Some of the more popular server-side scripting languages are PHP, ASP, and JSP. One of the primary reasons for using a server-side scripting language is to develop interactive Web sites that communicate with a database. Server-side scripting languages work in the processing tier and have the ability to handle communication between the client tier and the data storage tier. At the processing tier, a server-side scripting language usually prepares and processes the data in some way before submitting it to the data storage tier. Some of the more common uses of server-side scripting language that you have probably already seen on the Web include the following:

- Shopping carts
- Search engines
- Mailing lists and message boards
- Web-based e-mail systems
- Authentication and security mechanisms
- Web logs (blogs)
- Games and entertainment

Unlike JavaScript, a server-side scripting language can’t access or manipulate a Web browser. In fact, a server-side scripting language cannot run on a client tier at all. Instead, a server-side scripting language exists and executes solely on a Web server, where it performs various types of
processing or accesses databases. When a client requests a server-side script, the script is interpreted and executed by the scripting engine within the Web server software. After the script finishes executing, the Web server software then translates the results of the script (such as the result of a calculation or the records returned from a database) into HTML or XHTML, which it then returns to the client. In other words, a client will never see the server-side script, only the HTML or XHTML that the Web server software returns from the script. Figure 1-14 illustrates how a Web server processes a PHP script.

**Figure 1-14**
How a Web server processes a server-side script

---

### Should You Use Client-Side or Server-Side Scripting?

An important question in the design of any client/server system is deciding how much processing to place on the client or server. In the context of Web site development, you must decide whether to use client-side JavaScript or a server-side script. This is an important consideration because the choice you make can greatly affect the performance of your program. In some cases, the decision is simple. For example, if you want to control the Web browser, you must use JavaScript. If you want to access a database on a Web server, you must use a server-side script. However, there are tasks that both languages can accomplish, such as validating forms and manipulating cookies. Further, both languages can perform the same types of calculations and data processing.

A general rule of thumb is to allow the client to handle the user interface processing and light processing, such as data validation, but have the Web server perform intensive calculations and data storage. This division of labor is especially important when dealing with clients and servers over the Web. Unlike with clients on a private network, it’s not possible to know in advance the computing capabilities of each client on the Web. You cannot assume that each client (browser) that accesses your client/server application (Web site) has the necessary power to perform the processing required by the application. For this reason, intensive processing should be placed on the server.

Because servers are usually much more powerful than client computers, your first instinct might be to let the server handle all processing and only use the client to display a user interface. Although you do not want to overwhelm clients with processing they cannot handle, it is important to perform as much processing as possible on the client for several reasons:

- **Distributing processing among multiple clients creates applications that are more powerful, because the processing power is not limited to the capabilities of a single computer.** Client computers become more powerful every day, and advanced capabilities such as JavaScript are now available in local Web browsers. Thus, it makes sense to use a Web application to harness some of this power and capability. A **Web application** is a program that executes on a server but that clients access through a Web page loaded in a browser.
Local processing on client computers minimizes transfer times across the Internet and creates faster applications. If a client had to wait for all processing to be performed on the server, a Web application could be painfully slow over a busy Internet connection.

Performing processing on client computers lightens the processing load on the server. If all processing in a three-tier client/server system is on the server, the server for a popular Web site could become overwhelmed trying to process requests from numerous clients.

NOTE
The term “distributed application” is used to describe multiple computers sharing the computing responsibility for a single application.

The JavaScript Programming Language
The following sections introduce basic procedures for adding JavaScript to your Web pages.

The <script> Element
JavaScript programs run from within a Web page (either an HTML or XHTML document). That is, you type the code directly into the Web page code as a separate section. JavaScript programs contained within a Web page are often referred to as scripts. The <script> element tells the Web browser that the scripting engine must interpret the commands it contains. The type attribute of the <script> element tells the browser which scripting language and which version of the scripting language is being used. You assign a value of “text/javascript” to the type attribute to indicate that the script is written with JavaScript. You need to include the following code in a document to tell the Web browser that the statements that follow must be interpreted by the JavaScript scripting engine:

```html
<script type="text/javascript">
statements
</script>
```

Next, you will add a script section to the Forestville Funding page.

To add a script section to the Forestville Funding page:

1. Return to the ForestvilleFunding.html document in your text editor.
2. Add the following script section immediately after the <h2> element for the Fixed Rates heading:

```html
<script type="text/javascript">
</script>
```

3. Save the ForestvilleFunding.html document.
The individual lines of code, or statements, that make up a JavaScript program in a document are contained within the `<script>` element. The following script contains a single statement that writes the text “Bienvenue au Canada!” to a Web browser window, using the `write()` method of the `Document` object, which you will study shortly:

```javascript
document.write("<p>Bienvenue au Canada!</p>");
```

Notice that the preceding statement ends in a semicolon. Many programming languages, including C++ and Java, require you to end all statements with a semicolon. JavaScript statements are not required to end in semicolons. Semicolons are strictly necessary only when you want to separate statements that are placed on a single line. For example, the following script contains two statements on the same line, with each statement ending in a semicolon:

```javascript
<script type="text/javascript">
document.write("<p>Bienvenue ");
document.write("au Canada!</p>");
</script>
```

As long as you place each statement on its own line, separated from other lines with line breaks, you are not required to end statements with semicolons. The following code shows another example of the preceding script, but this time, each statement is placed on its own line, without an ending semicolon.

```javascript
<script type="text/javascript">
document.write("<p>Bienvenue ");
document.write("au Canada!</p>");
</script>
```

Even though the statements do not end in semicolons, the preceding script is legal. However, that’s not the end of the story. Programmers often adopt conventions in their code that make the code easier for the programmer to read in a text editor. In the case of semicolons, it is considered good JavaScript programming practice to end any statement with a semicolon. The semicolon serves to identify the end of each statement, making it easier for the programmer to read his or her own code (and for other programmers to read the code later on). Therefore, be sure to end all of your JavaScript statements with semicolons.

**NOTE**

Although this book covers JavaScript, you can also use other scripting languages with Web pages. To use VBScript in your Web pages, you would use the following code: `<script type="text/vbscript">VBScript code </script>`. Do not confuse JScript with VBScript. JScript is Microsoft’s version of the JavaScript scripting language. To specify the JScript language, you specify JavaScript as the type attribute.

If you anticipate that your JavaScript programs will run only in Internet Explorer, then you can specify “JScript” as your scripting language by using the statement `<script type="JScript">`. However, few browsers other than Internet Explorer will recognize “JScript” as a valid type attribute for the `<script>` element; it is safer always to use “JavaScript.”
HTML documents use the `language` attribute to tell the browser which scripting language and which version of the scripting language is being used. However, the `language` attribute is deprecated, so be sure to use the `type` attribute with your XHTML documents.

**Understanding JavaScript Objects**

Before you can use `<script>` elements to create a JavaScript program, you need to learn some basic terminology that is commonly used in JavaScript programming and in other kinds of programming languages. In addition to being an interpreted scripting language, JavaScript is considered an object-based programming language. An object is programming code and data that can be treated as an individual unit or component. For example, you might create a `Loan` object that calculates the number of payments required to pay off a loan. The `Loan` object may also store information such as the principal loan amount and the interest rate. Individual statements used in a computer program are often grouped into logical units called procedures, which are used to perform specific tasks. For example, a procedure may contain a group of statements that calculate the sales tax based on sales total. The procedures associated with an object are called methods.

A property is a piece of data, such as a color or a name, that is associated with an object. In the `Loan` object example, the programming code that calculates the number of payments required to pay off the loan is a method. The principal loan amount and the interest rate are properties of the `Loan` object.

To incorporate an object and an associated method in JavaScript code, you type the object’s name, followed by a period, followed by the method. For example, the following code shows the `Loan` object, followed by a period, followed by a method named `calcPayments()`, which calculates the number of payments required to pay off the loan:

```javascript
loan.calcPayments();
```

For many methods, you also need to provide some more specific information, called an argument, between the parentheses. Some methods require numerous arguments, while others don’t require any. Providing an argument for a method is referred to as passing arguments. For example, the `calcPayments()` method may require an argument that specifies the amount paid each month toward the loan. In that case, the JavaScript statement would look like this:

```javascript
loan.calcPayments(800);
```

You use an object’s properties in much the same way you use a method, by appending the property name to the object with a period. However, a property name is not followed by parentheses. One of the biggest differences between methods and properties is that a property does not actually do anything; you only use properties to store data. You assign a value to a property using an equal sign, as in the following example:

```javascript
loan.interest = .08;
```

The next part of this chapter focuses on the `write()` and `writeln()` methods as a way of helping you understand how to program with JavaScript.
Using the `write()` and `writeln()` Methods

JavaScript treats many things as objects. One of the most commonly used objects in JavaScript programming is the `Document` object. The `Document` object represents the content of a browser’s window. Any text, graphics, or other information displayed in a Web page is part of the `Document` object. One of the most common uses of the `Document` object is to add new text to a Web page. You create new text on a Web page with the `write()` method or the `writeln()` method of the `Document` object. For example, you could use the `write()` method to render a Web page containing custom information such as a user’s name or the result of a calculation.

You should understand that the only reason to use the `write()` and `writeln()` methods is to add new text to a Web page while it is being rendered. For example, you may want to display a new Web page based on information a user enters into a form. A user may enter, say, sales information into a form for an online transaction. Using the entered information, you can create a new Web page that displays their sales total, order confirmation, and so on. If you simply want to display text in a Web browser when the document is first rendered, there is no need to use anything but standard XHTML elements. The procedures for dynamically gathering information are a little too complicated for this introductory chapter. However, in this chapter you will use the `write()` and `writeln()` methods to display text in a Web browser when the document is first rendered in order to learn the basics of creating JavaScript programs.

Different methods require different kinds of arguments. For example, the `write()` and `writeln()` methods of the `Document` object require a text string as an argument. A text string, or literal string, is text that is contained within double or single quotation marks. The text string argument of the `write()` and `writeln()` methods specifies the text that the `Document` object uses to create new text on a Web page. For example, `document.write("Bienvenue au Canada!");` displays the text “Bienvenue au Canada!” in the Web browser window (without the quotation marks). Note that you must place literal strings on a single line. If you include a line break within a literal string, you receive an error message.

If you are using a version of Internet Explorer higher than 4, you need to turn on error notification by selecting Internet Options from the Tools menu and clicking the Advanced tab. In the Browsing category on the Advanced tab, make sure the Display a notification about every script error check box is selected, and click the OK button to close the dialog box. To view errors in Firefox, you must select Error Console from the Tools menu.

**TIP**

Programmers often talk about code that “writes to” or “prints to” a Web browser window. For example, you might say that a piece of code writes a text string to the Web browser window. This is just another way of saying that the code displays the text string in the Web browser window.

The `write()` and `writeln()` methods perform essentially the same function that you perform when you manually add text to the body of a standard Web page document. Whether you add text to a document by using standard elements such as the `<p>` element or by using the `write()` or `writeln()` methods, the text is added according to the order in which the statement appears in the document.
The only difference between the `write()` and `writeln()` methods is that the `writeln()` method adds a line break after the line of text. Line breaks, however, are only recognized inside the `<pre>` element. In other words, in order to use line breaks with the `writeln()` method, you must place the method within a `<pre>` element. The following code contains a script that prints some text in a Web browser using the `writeln()` method of the `Document` object. Notice that the `<script>` element is enclosed within a `<pre>` element. Figure 1-15 shows the output.

```html
<pre style="color: blue; font-family: Arial; font-size: .8em; font-weight: normal">
<script type="text/javascript">
document.writeln("Abraham Lincoln once said:");
document.writeln("<em>Tact is the ability to describe others as they see themselves.";)
</script>
</pre>

Note the use of semicolons at the end of each statement. Remember that it is considered good JavaScript programming practice to end any statement with a semicolon.

**Figure 1-15** Output of a script that uses the `writeln()` method of the `Document` object

Notice that the second `writeln()` statement includes the XHTML element `<em>`. You can include any elements you like as part of an argument for the `write()` or `writeln()` methods, including elements such as the `<p>` and `<br>` elements. This means that you can use `write()` statements to add line breaks to the text you create with a script instead of using `writeln()` statements within a `<pre>` element. The following code shows a modified version of the previous script, but this time it uses `write()` statements and does not include a `<pre>` element. The line break in the text is created by adding a `<br />` element to the end of the first line of text.

```html
<script type="text/javascript">
document.write("Abraham Lincoln once said:<br />");
document.write("<em>Tact is the ability to describe others as they see themselves.";)
</script>

Next, you will modify the elements and text in the Fixed Rates heading so they are added using `write()` methods of the `Document` object.
To modify the elements and text in the Fixed Rates heading so they are added using `write()` methods of the `Document` object:

1. Return to the `ForestvilleFunding.html` document in your text editor.
2. Add to the script section the following `document.write()` statements which print the text and elements that should appear beneath the Fixed Rates heading:

   ```javascript
   document.write("<ul>;
   document.write("<li>30-year: 6.25%</li>);
   document.write("<li>15-year: 6.00%</li>);
   document.write("</ul>);
   ```

   Remember that the only reason you are using `write()` statements to add text to a Web page when it is first rendered is to learn the basics of JavaScript.

3. Now delete the original text elements located between the closing `</script>` tag and the `<h2>` element containing “Adjustable Rates”.
4. Save the `ForestvilleFunding.html` document and open it in your Web browser. The document should appear the same as it did before you added the script section.
5. Close your Web browser window.

**Case Sensitivity in JavaScript**

Like XHTML, JavaScript is case sensitive, and within JavaScript code, object names must always be all lower case. This can be a source of some confusion, because in written explanations about JavaScript, the names of objects are usually referred to with an initial capital letter. For example, throughout this book, the `Document` object is referred to with an uppercase D. However, you must use a lowercase d when referring to the `Document` object in a script. The statement

```javascript
Document.write("Bienvenue au Canada!");
```

causes an error message because the JavaScript interpreter cannot recognize an object named `Document` with an uppercase D.

Similarly, the following statements will also cause errors:

```javascript
DOCUMENT.write("Bienvenue au Canada!");
Document.Write("Bienvenue au Canada!");
document.WRITE("Bienvenue au Canada!");
```

**Adding Comments to a JavaScript Program**

When you create a program, whether in JavaScript or any other programming language, it is considered good programming practice to add comments to your code. In this section, you will learn how to create JavaScript comments. Comments are nonprinting lines that you place in your code to contain various types of remarks, including the name of the program, your name and the date you created the program, notes to yourself, or instructions to future programmers who may need to modify your work. When you are working with long scripts, comments make it easier to decipher how a program is structured.

JavaScript supports two kinds of comments: line comments and block comments. A **line comment** hides a single line of code. To create a line comment, add two slashes `//` before the text you want to use as a comment. The `//` characters instruct the JavaScript interpreter to ignore all
text immediately following the slashes to the end of the line. You can place a line comment either at the end of a line of code or on its own line. **Block comments** hide multiple lines of code. You create a block comment by adding /* to the first line that you want included in the block, and you close a comment block by typing */ after the last character in the block. Any text or lines between the opening /* characters and the closing */ characters are ignored by the JavaScript interpreter. The following code shows a <script> element containing line and block comments. If you open a document that contains the following script in a Web browser, the browser does not render the text marked with comments.

```html
<script type="text/javascript">
/*
This line is part of the block comment. This line is also part of the block comment.
*/
document.writeln("<h1>Comments Example</h1>"); // Line comments can follow code statements
// This line comment takes up an entire line.
/* This is another way of creating a block comment. */
</script>
```

**TIP**
Comments in JavaScript use the same syntax as comments created in C++, Java, and other programming languages.

Next, you will add comments to the Forestville Funding page.

**To add comments to the Forestville Funding page:**

1. Return to the **ForestvilleFunding.html** document in your text editor.
2. Add the following block comment immediately after the opening <script> tag:
   
   ```javascript
   /*
   JavaScript code for Chapter 1.
   The purpose of this code is simply to demonstrate how to add a script section to a Web page.
   */
   ```
3. Next, add the following line comments immediately after the block comment, taking care to replace “your name” with your first and last name and “today’s date” with the current date:
   
   ```javascript
   // your name
   // today’s date
   ```
4. Save the **ForestvilleFunding.html** document, and then open it in your Web browser to confirm that the comments are not displayed.
5. Close your Web browser window.
Structuring JavaScript Code

When you add JavaScript code to a document, you need to follow certain rules regarding the placement and organization of that code. The following sections describe some important rules to keep in mind when structuring your JavaScript code.

Including a `<script>` Element for Each Code Section

You can include as many script sections as you like within a document. However, when you include multiple script sections in a document, you must include a `<script>` element for each section. The following document includes two separate script sections. The script sections create the information that is displayed beneath the `<h2>` heading elements. Figure 1-16 shows the output.

```html
<h1>Multiple Script Sections</h1>
<h2>First Script Section</h2>
<script type="text/javascript">
  document.write("<p>Output from the first script section.</p>"
);
</script>
<h2>Second Script Section</h2>
<script type="text/javascript">
  document.write("<p>Output from the second script section.</p>"
);
</script>
```

Figure 1-16
Output of a document with two JavaScript sections

Next, you will modify the text and comments beneath the “Adjustable Rates” and “Popular Mortage Programs” headings so they are added to the page from a script section.

To modify the text and comments beneath the “Adjustable Rates” and “Popular Mortage Programs” headings so they are added to the page from a script section:

1. Return to the `ForestvilleFunding.html` document in your text editor.
2. Modify the text and elements beneath the “Adjustable Rates” heading so they are added to the page from a script section, as follows:

```html
<script type="text/javascript">
  document.write("<ul>");
</script>
```
3. Modify the text and elements beneath the “Popular Mortgage Programs” heading so they are added to the page from a script section, as follows.

```html
<script type="text/javascript">
    document.write("<ul>");
    document.write("<li>No money down</li>);
    document.write("<li>No income or asset verification</li>);
    document.write("<li>Interest only loans</li>);
    document.write("<li>Home equity loans</li>");
    document.write("</ul>");
</script>
```

4. Save the `ForestvilleFunding.html` document and open it in your Web browser. The document should appear the same as it did before you added the new script sections.

5. Close your Web browser window.

### Placing JavaScript in the Document Head or Document Body

You can place `<script>` elements in either the document head or document body. Where you place your `<script>` elements varies, depending on the program you are writing. The statements in a script are rendered in the order in which they appear in the document. As a general rule, then, it is a good idea to place as much of your JavaScript code as possible in the document head, because the head of a document is rendered before the document body. When placed in the document head, JavaScript code is processed before the main body of the document is displayed. It is especially important to place JavaScript code in the document head when your code performs behind-the-scenes tasks that are required by script sections located in the document body.

Next, you will modify the `<h1>` element and the `<hr>` element that follows it so they are added to the page from a script section in the document head.

To modify the `<h1>` element and the `<hr>` element that follows it so they are added to the page from a script section in the document head:

1. Return to the `ForestvilleFunding.html` document in your text editor.
2. Create a new script section immediately above the closing `</head>` tag, as follows:

```html
<script type="text/javascript">
</script>
```

3. Add the following `document.write()` statement to the new script section, which prints the `<h1>` and `<hr>` elements:

```html
    document.write("<h1>Forestville Funding</h1><hr />");
```

4. Delete the `<h1>` and `<hr>` elements that follow the opening `<body>` tag.
5. Save the **ForestvilleFunding.html** document and open it in your Web browser. The document should appear the same as it did before you added the new script sections.


**Creating a JavaScript Source File**

JavaScript is often incorporated directly into a Web page. However, you can also save JavaScript code in an external file called a JavaScript source file. You can then write a statement in the document that executes (or “calls”) the code saved in the source file. When a browser encounters a line calling a JavaScript source file, it looks in the JavaScript source file and executes it.

A JavaScript source file is usually designated by the file extension .js and contains only JavaScript statements; it does not contain a `<script>` element. Instead, the `<script>` element is located within the document that calls the source file. To access JavaScript code that is saved in an external file, you use the `src` attribute of the `<script>` element. You assign to the `src` attribute the Uniform Resource Locator (URL) of a JavaScript source file. For example, to call a JavaScript source file named scripts.js, you would include the following code in a document:

```html
<script type="text/javascript" src="scripts.js"></script>
```

JavaScript source files cannot include XHTML elements. If you include XHTML elements in a JavaScript source file, your Web page generates an error message. Also, when you specify a source file in your document using the `src` attribute, the browser ignores any other JavaScript code located within the `<script>` element. For example, consider the following JavaScript code. In this case, the JavaScript source file specified by the `src` attribute of the `<script>` element executes properly, but the `write()` statement is ignored.

```html
<script type="text/javascript" src="scripts.js"> 
    document.write("<p>This statement will be ignored.</p>"); 
</script>
```

If the JavaScript code you intend to use in a document is fairly short, then it is usually easier to include JavaScript code in a `<script>` element within the document itself. However, for longer JavaScript code, it is easier to include the code in a .js source file. There are several reasons you may want to use a .js source file instead of adding the code directly to a document:

- Your document will be neater. Lengthy JavaScript code in a document can be confusing. You may not be able to tell at a glance where the XHTML code ends and the JavaScript code begins.
- The JavaScript code can be shared among multiple Web pages. For example, your Web site may contain pages that allow users to order an item. Each Web page displays a different item but uses the same JavaScript code to gather order information. Instead of recreating the JavaScript order information code within each document, the Web pages can share a central JavaScript source file. Sharing a single source file among multiple documents reduces disk space. In addition, when you share a source file among multiple documents, a Web browser needs to keep only one copy of the file in memory, which reduces system overhead.
JavaScript source files hide JavaScript code from incompatible browsers. If your document contains JavaScript code, an incompatible browser displays that code as if it were standard text. By contrast, if you put your code in a source file, incompatible browsers simply ignore it.

You can use a combination of embedded JavaScript code and JavaScript source files in your documents. The ability to combine embedded JavaScript code and JavaScript source files in a single Web page is advantageous if you have multiple Web pages, each of which requires individual JavaScript code statements, but all of which also share a single JavaScript source file.

Suppose you have a Web site with multiple Web pages. Each page displays a product that your company sells. You may have a JavaScript source file that collects order information, such as a person’s name and address, that is shared by each of the product Web pages. Each individual product page may also require other kinds of order information that you need to collect using JavaScript code. For example, one of your products may be a shirt, for which you need to collect size and color information. On another Web page, you may sell jellybeans, for which you need to collect quantity and flavor information. Each of these products can share a central JavaScript source file to collect standard information, but each may also include embedded JavaScript code to collect product-specific information.

Next, you will create a JavaScript source file that prints the contents of the “Popular Mortgage Programs” heading.

To create a JavaScript source file that prints the contents of the “Popular Mortgage Programs” heading:

1. Create a new document in your text editor.
2. Add the following statements, which print the text that will be displayed beneath the “Popular Mortgage Programs” heading. These will be the only statements in the document. Remember that you do not include a script section within a source file.

   ```javascript
   document.write("<ul>");
   document.write("<li>No money down</li>");
   document.write("<li>No income or asset verification</li>");
   document.write("<li>Interest only loans</li>");
   document.write("<li>Home equity loans</li>");
   document.write("</ul>");
   ```
3. Save the document as mortgage.js in the Chapter folder for Chapter 1 and then close it.
4. Now return to the ForestvilleFunding.html document in your text editor.
5. Delete the statements that appear within the script section for the “Popular Mortgage Programs” heading.
6. Add an src attribute to the opening <script> tag for the “Popular Mortgage Programs” heading as follows, so it calls the external JavaScript source file:

   ```html
   <script type="text/javascript" src="mortgage.js">
   </script>
   ```
7. Save the `ForestvilleFunding.html` document and open it in your Web browser. The document should appear the same as it did before you added the new script sections.

8. Close your Web browser window.

**Writing Valid JavaScript Code**

You should always strive to create Web pages that conform to the rules and requirements of XHTML. However, JavaScript can prevent an XHTML document from being well formed because some JavaScript statements contain symbols such as the less-than symbol (<) symbol, the greater-than symbol (>), and the ampersand (&). This is not a problem with HTML documents because the statements in a `<script>` element are interpreted as character data instead of as markup. A section of a document that is not interpreted as markup is referred to as character data, or CDATA. If you were to validate an HTML document containing a script section, the document would validate successfully because the validator would ignore the script section and not attempt to interpret the text and symbols in the JavaScript statements as HTML elements or attributes. By contrast, with XHTML documents, the statements in a `<script>` element are treated as parsed character data, or PCDATA, which identifies a section of a document that is interpreted as markup.

JavaScript code in an XHTML document is treated as PCDATA. That means that if you attempt to validate an XHTML document that contains a script section, it will fail the validation. To avoid this problem, you can do one of two things. One option is to move your code into a source file, which prevents the validator from attempting to parse the JavaScript statements. Alternatively, if you prefer to keep the JavaScript code within the document, you can enclose the code within a `<script>` element within a CDATA section, which marks sections of a document as CDATA. The syntax for including a CDATA section on a Web page is as follows:

```html
<![[CDATA[
statements to mark as CDATA
]]>}
```

The following example contains JavaScript code that is enclosed within a CDATA section.

```html
<body>
<script type="text/javascript">
<![CDATA[
document.write("<h1>Order Confirmation</h1>");
document.write("<p>Your order has been received.</p>");
document.write("<p>Thank you for your business!</p>");
]]>
</script>
</body>
</html>
```

The CDATA section in the preceding example prevents the validator from attempting to parse the JavaScript statements. However, the JavaScript interpreter will attempt to treat the `<![CDATA[ and ]]>` lines as JavaScript statements, which will cause an error. To avoid this...
problem, you can enclose the opening and closing portions of a CDATA section within block comments, as shown in the following example. Figure 1-17 shows the output.

```html
<body>
<script type="text/javascript">
/* <![CDATA[ */
document.write("<h1>Order Confirmation</h1>");
document.write("<p>Your order has been received.</p>");
document.write("<p>Thank you for your business!</p>");
/* ]]> */
</script>
</body>
</html>
```

**Figure 1-17**
Output of Web page with hidden JavaScript code

Next, you will modify the script sections in the Forestville Funding page so they are hidden from incompatible browsers and are well formed.

To modify the script sections in the Forestville Funding page so they are hidden from incompatible browsers and are well formed:

1. Return to the *ForestvilleFunding.html* document in your text editor.
2. Modify the script section in the document head (that prints the `<h1>` and `<hr>` elements) as follows so the code is contained with a CDATA section to ensure that the Web page can be validated:

```html
<script type="text/javascript">
/* <![CDATA[ */
document.write("<h1>Forestville Funding</h1><hr />");
/* ]]> */
</script>
```
3. Add the same code to the script sections for the Fixed Rates and Adjustable Rates headings. (You do not need to add the statements to the “Popular Mortgage Programs” heading because it calls an external JavaScript source file.)
4. Save the *ForestvilleFunding.html* document, validate it with the W3C Markup Validation Service, and open it in your Web browser. You should be using a browser that supports JavaScript, so the document should be rendered correctly.

5. Close your Web browser window and text editor window.

## Logic and Debugging

All programming languages, including JavaScript, have their own **syntax**, or rules. To write a program, you must understand the syntax of the programming language you are using. You must also understand computer-programming logic. The term **logic** refers to the order in which various parts of a program run, or execute. The statements in a program must execute in the correct order to produce the desired results. In an analogous situation, although you know how to drive a car well, you may not reach your destination if you do not follow the correct route. Similarly, you might be able to write statements using the correct syntax, but be unable to construct an entire, logically executed program that works the way you want. Similarly, you might be able to use a programming language’s syntax correctly, but be unable to execute a logically constructed, workable program. A typical logical error might be multiplying two values when you meant to divide them. Another might be producing output before obtaining the appropriate input (for example, printing an order confirmation on the screen before asking the user to enter the necessary order information).

Any error in a program that causes it to function incorrectly, whether because of incorrect syntax or flaws in logic, is called a **bug**. The term **debugging** refers to the act of tracing and resolving errors in a program. Grace Murray Hopper, a mathematician who was instrumental in developing the Common Business-Oriented Language (COBOL) programming language, is said to have first coined the term “debugging.” As the story from the 1940s goes, a moth short-circuited a primitive computer that Hopper was using. Removing the moth from the computer “debugged” the system and resolved the problem. Today, a bug refers to any sort of problem in the design and operation of a program.

---

**CAUTION**

Do not confuse bugs with computer viruses. Bugs are problems within a program that occur because of syntax errors, design flaws, or run-time errors. Viruses are self-contained programs designed to “infect” a computer system and cause mischievous or malicious damage. Virus programs themselves can contain bugs if they contain syntax errors or do not perform as their creators envisioned.

**NOTE**

As you work through this book, keep in mind that debugging is not an exact science. Every program you write is different and requires different methods of debugging. While there are some tools available to help you debug your JavaScript code, your own logical and analytical skills are the best debugging resources you have.
Chapter Summary

- In 1990 and 1991, Tim Berners-Lee created what would become the World Wide Web, or the Web, at the European Laboratory for Particle Physics (CERN) in Geneva, Switzerland, as a way to easily access cross-referenced documents that existed on the CERN computer network.
- The World Wide Web Consortium, or W3C, was established in 1994 at the Massachusetts Institute of Technology (MIT) to oversee the development of Web technology standards.
- HTML documents are text documents that contain formatting instructions, called tags, which determine how data is displayed on a Web page. A tag pair and the data it contains are referred to as an element.
- Extensible Hypertext Markup Language, or XHTML, is the next generation markup language for creating the Web pages that appear on the World Wide Web.
- A document type definition, or DTD, defines the elements and attributes that can be used in a document, along with the rules that a document must follow when it includes them.
- To design and format the display of Web pages for traditional Web browsers, you use CSS, a standard set by the W3C for managing the design and formatting of Web pages in a Web browser.
- A validating parser is a program that checks whether a document is well formed and whether the document conforms to a specific DTD.
- Web page design, or Web design, refers to the visual design and creation of the documents that appear on the World Wide Web.
- Web page authoring (or Web authoring) is the process of creating and assembling the tags, attributes, and data that make up a Web page.
- The term “Web development” (or “Web programming”) refers to the design of software applications for a Web site.
- In traditional client/server architecture, the server is usually some sort of database from which a client requests information.
- A system consisting of a client and a server is known as a two-tier system. The Web is built on a two-tier client/server system, in which a Web browser (the client) requests documents from a Web server.
- A three-tier, or multitier, client/server system consists of three distinct pieces: the client tier, the processing tier, and the data storage tier.
- JavaScript is a client-side scripting language that allows Web page authors to develop interactive Web pages and sites.
- An object is programming code and data that can be treated as an individual unit or component.
- Comments are nonprinting lines that you place in your code to contain various types of remarks, including the name of the program, your name and the date you created the program, notes to yourself, or instructions to future programmers who may need to modify your work.
- Character data, or CDATA, refers to a section of a document that is not interpreted as markup.
- Parsed character data, or PCDATA, identifies section of a document that should be parsed.
To ensure that you can validate an XHTML document that contains a script section, you can move code into a source file or enclose the code within a `<script>` element within a CDATA section.

The term “logic” refers to the order in which various parts of a program run, or execute.

Any error in a program that causes it to function incorrectly, whether because of incorrect syntax or flaws in logic, is called a bug.

The term “debugging” refers to the act of tracing and resolving errors in a program.

Review Questions

1. Why were the Internet and World Wide Web developed?
2. Hypertext linking was originally developed as a way to easily access cross-referenced documents that were stored on the CERN computer network. True or false?
3. Which of the following is a generic term for many types of names and addresses on the World Wide Web?
   a. URM
   b. URI
   c. URA
   d. URL
4. Which organization created the first program to allow users to navigate the Web using a graphical user interface (GUI)?
   a. Microsoft
   b. Netscape
   c. World Wide Web Consortium (W3C)
   d. University of Illinois
5. Netscape and Firefox are currently the most popular browsers on the market. True or false?
6. What are the basic elements that define the structure of a Web page? What are each of these elements called?
7. The process by which a Web browser assembles or formats an HTML document is called ______. (Choose all that apply.)
   a. parsing
   b. compiling
   c. rendering
   d. interpreting
8. Most of today’s Web browsers can be used to create Web pages. True or false?
9. What is a user agent and why should you use XHTML with them?
10. Which XHTML DTD(s) allow you to use deprecated elements? (Choose all that apply.)
   a. XML
   b. transitional
   c. strict
   d. frameset

11. Explain the basic rules for writing well-formed documents.

12. Which element do you use to create an internal style sheet?
   a. <style>
   b. <styles>
   c. <css>
   d. <link>

13. Explain when you should use inline styles, internal style sheets, or external style sheets.

14. If you do not validate an XHTML document and it contains errors, most Web browsers will not render the page. True or false?

15. Explain the difference between Web page authoring and Web development.

16. The _____ tier in a three-tier client/server system handles the interaction between the Web browser client and the data storage tier.
   a. client
   b. processing
   c. data storage
   d. rendering

17. Explain the difference between server-side scripting and client-side scripting.

18. JavaScript statements must end in semicolons. True or false?

19. Which of the following statements causes an error? (Choose all that apply.)
   a. Document.write("There's no business like show business!");
   b. document.Write("There's no business like show business!");
   c. Document.Write("There's no business like show business!");
   d. document.write("There's no business like show business!");

20. Explain how to ensure that an XHTML document containing JavaScript can be validated.
Hands-On Projects

Project 1-1

In this project, you will create a Web page that displays information about the country of Azerbaijan. The Web page will be an HTML document that does not conform to XHTML.

1. Create a new document in your text editor, and type the `<html>` element, document head, and `<body>` element. Use “Azerbaijan” as the content of the `<title>` element. Your document should appear as follows:

```html
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Azerbaijan</title>
<meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />
</head>
<body>
</body>
</html>
```

2. Next, add the following text and elements to the document body. Notice that the document includes the deprecated `<center>` and `<font>` elements.

```html
<center><font face="arial" color="navy">
<h1>Azerbaijan</h1>
<p><b>Official name</b>: <i>Republic of Azerbaijan</i><br />
<b>Ethnic groups</b>: <i>Azeri 90%, Dagestani 3%, Russian 3%, Armenian 2%</i><br />
<b>Principal languages</b>: <i>Azeri (official), Russian, Armenian</i><br /></p>
</font></center>
```

3. Save the document as `Azerbaijan.html` in the Projects folder for Chapter 1, then open it in your Web browser and examine how the elements are rendered.


5. Change the `<i>` and `<b>` elements in the document body to the `<em>` and `<strong>` phrase elements. The document body should appear as follows:

```html
<center><font face="arial" color="navy">
<h1>Azerbaijan</h1>
<p><strong>Official name</strong>: <em>Republic of Azerbaijan</em><br />
<strong>Ethnic groups</strong>: <em>Azeri 90%, Dagestani 3%, Russian 3%, Armenian 2%</em><br />
</font></center>
```
6. Save the document as **Azerbaijan.html** document, close it in your text editor, and then open it in your Web browser and examine how the elements are rendered. The document should appear the same as before you changed the `<i>` and `<b>` elements to the `<em>` and `<strong>` phrase elements.

7. Close your Web browser window.

**Project 1-2**

In this project, you will create a Web page that displays ultraviolet index forecast information. The Web page will conform to the strict DTD.

1. Create a new document in your text editor, and type the `<html>` element, document head, and `<body>` element. Use “Ultraviolet (UV) Index Forecast” as the content of the `<title>` element. Your document should appear as follows:

```html
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Ultraviolet (UV) Index Forecast</title>
<meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />
</head>
<body>
</body>
</html>
```

2. Add the following text and elements to the document body. The elements build a table that display the ultraviolet index forecast information.

```html
<h1>Ultraviolet (UV) Index Forecast</h1>
<table width="100%" border="1">
<tr align="left"><th>UV Index</th><th>Exposure</th><th>Minimum Precautions</th></tr>
<tr><td>0-2</td><td>Minimal</td><td>SPF 15 sun screen</td></tr>
<tr><td>3-4</td><td>Low</td><td>Sun screen and hat</td></tr>
<tr><td>5-6</td><td>Moderate</td><td>Sun screen, hat, UV sunglasses</td></tr>
<tr><td>7-9</td><td>High</td><td>Sun screen, hat, UV sunglasses, avoid sun between 10am and 4am</td></tr>
</table>
```

3. Save the document as **UVIndex.html** in the Projects folder for Chapter 1.
Use the W3C Markup Validation Service to validate the UVIndex.html document and fix any errors that the document contains. Once the document is valid, close it in your text editor and then open it in your Web browser and examine how the elements are rendered.

Close your Web browser window.

### Project 1-3

In this project, you will create a Web page that describes the Super Bowl and that contains inline styles. The Web page will conform to the strict DTD.

1. Create a new document in your text editor, and type the `<html>` element, document head, and `<body>` element. Use “Super Bowl” as the content of the `<title>` element. Your document should appear as follows:

```html
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Super Bowl</title>
<meta http-equiv="content-type" content="text/html; charset=iso-8859-1" />
</head>
<body>
</body>
</html>
```

2. Modify the opening `<body>` tag so it contains inline styles as follows:

```html
<body style="color: black; background: white; margin-bottom: 6ex; font-family: Verdana, Arial, Helvetica, sans-serif; font-size: 12px">
```

3. Add the following heading element to the document body. The `<h1>` element contains a `style` attribute that defines several inline styles for the heading.

```html
<h1 style="color: #039; background: white; margin-bottom: 2ex">Super Bowl</h1>
```

4. Add the following paragraph to the document body. The `<p>` element also contains a `style` attribute that defines several inline styles for the heading.

```html
<p style="margin-left: 6em;margin-right: 6em">The Super Bowl is the end-of-season championship game in American football. This game takes place between the winners of the two major American leagues: the National Football Conference (NFC) and the American Football Conference (AFC). The first Super Bowl took place in 1967 and now takes place in January or February following the regular season.</p>
```
5. Save the document as **SuperBowlInline.html** in the Projects folder for Chapter 1, and then open it in your Web browser and examine how the elements and styles are rendered.

6. Close your Web browser window, but leave the SuperBowlInline.html document open in your text editor.

**Project 1-4**

In this project, you will modify the Super Bowl Web page so it uses an internal style sheet instead of inline styles.

1. Return to the **SuperBowlInline.html** document in your text editor and immediately save it as **SuperBowlInternal.html** in the Projects folder for Chapter 1.

2. Add the following internal style sheet to the document head, immediately above the `</head>` tag. This style sheet contains the same inline styles that you added in the last exercise.

   ```html
   <style type="text/css">
   body {
   color: black;
   background: white;
   margin-bottom: 6ex;
   font-family: Verdana, Arial, Helvetica, sans-serif;
   font-size: 12px;
   }
   h1 {
   color: #039;
   background: white;
   margin-bottom: 2ex;
   }
   p {
   margin-left: 6em;
   margin-right: 6em;
   }
   </style>
   
   3. Delete the inline styles from the `<h1>`, `<body>`, and `<p>` tags.

4. Save the **SuperBowlInternal.html** document and open it in your Web browser. The elements should appear the same as they did before you converted the inline styles to an internal style sheet.

5. Close your Web browser window, but leave the SuperbowlInternal.html document open in your text editor.

**Project 1-5**

In this project, you will modify the Super Bowl Web page so it uses an external style sheet instead of an internal style sheet.
1. Return to the SuperBowlInternal.html document in your text editor and immediately save it as SuperBowlExternal.html in the Projects folder for Chapter 1.

2. Copy the style declarations to your Clipboard, but be sure not to copy the <style> or </style> tags.

3. Create a new document in your text editor and paste the style declarations from your Clipboard.

4. Save the document as superbowl_styles.css in the Projects folder for Chapter 1, and then close it in your text editor.

5. Return to the SuperBowlExternal.html document in your text editor and delete the <style> element and the declarations it contains.

6. Add the following <link> element to the document head, immediately above the </head> tag. This element links to the external superbowl_styles.css style sheet.

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

7. Save the SuperBowlExternal.html document and open it in your Web browser.

8. Use the W3C Markup Validation Service to validate the SuperBowlExternal.html document and fix any errors that the document contains. Once the document is valid, close it in your text editor and then open it in your Web browser and examine how the elements are rendered. The elements should appear the same as they did before you converted the internal style sheet to an external style sheet.


---

**Project 1-6**

In this project you will create a Web page that uses document.write() statements in a script section to print dietary recommendations for a healthy heart. The Web page will conform to the strict DTD.

1. Create a new document in your text editor, and type the <html> element, document head, and <body> element. Use “Dietary Recommendations” as the content of the <title> element. Your document should appear as follows:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Dietary Recommendations</title>
<meta http-equiv="content-type" content="text/html;
charset=iso-8859-1" />
</head>
<body>
</body>
</html>
```
2. Add the following text and elements to the document body:

```html
<h1>Dietary Recommendations</h1>
<p>The American Heart Association recommends the following dietary guidelines for a healthy heart:</p>
```

3. Add the following script section to the end of the document body. The script section contains a CDATA section to ensure that the Web page is valid and includes block comments that will contain your name, the current date, and “Project 1-6”. Be sure to add your name and the current date where indicated.

```html
<script type="text/javascript">
/* <![CDATA[ */
/*
 your name
 current date
 Project 1-6
 */
/* ]]> */
</script>
```

4. Add the following `document.write()` statements to the script section immediately after the statement containing the closing block comment characters (`*/`). These statements use an unordered list element to print dietary recommendations for a healthy heart.

```html
document.write("<ul>");
document.write("<li>Eat less fat</li>"

document.write("<li>Avoid sugary and processed foods</li>"

document.write("<li>Eat plenty of fiber-rich foods</li>"

document.write("<li>Cut down on salt</li>"

document.write("<li>Eat at least 400g of fruit and vegetables each day</li>"

document.write("<ul>");
```

5. Save the document as `HealthyHeart.html` in the Projects folder for Chapter 1.

6. Use the W3C Markup Validation Service to validate the `HealthyHeart.html` document and fix any errors that the document contains. Once the document is valid, close it in your text editor and then open it in your Web browser and examine how the elements are rendered.

7. Close your Web browser window.
Case Projects

Save your Case Projects files in the Cases folder for Chapter 1. Be sure to validate the files you create with the W3C Markup Validation Service.

**Case Project 1-1**

Create a Web page for a store that rents computers by the hour. Use the strict DTD and an internal style sheet. Format the heading level styles in olive and the paragraphs in blue. Format the heading and body elements using sans-serif fonts such as Arial and Helvetica. Include headings such as Services Offered, Hours of Operation, Rental Charges, and Accepted Forms of Payment. Within the Rental Charges heading, create a table that lists the cost of different types of computer platforms, such as Windows, Linux, and Macintosh. Format the rows in the table so they alternate from white to gray. Within the gray rows, format the text to be white. Within the white rows, format the text to be black. You will need to set the `color` and `background-color` properties for the table’s `<tr>` elements using class selectors. Save the Web page as `ComputerCenter.html`.

**Case Project 1-2**

Create a Web page for a company that rents horses. Use the strict DTD and an external style sheet. Format the heading elements in navy and the paragraphs in black. Use the body selector to format all of the text in the body using serif fonts such as Garamond and Times New Roman. Use whatever size you like for the heading and paragraph font sizes. Include at least three paragraphs that describe the services the company offers. Format each paragraph so its line height is spaced at 150%. Also, format the first word in every paragraph so it is 30% larger than the surrounding text, formatted in blue, and uses a sans-serif font such as Arial. Save the Web page as `HorseRentals.html` and the style sheet as `horses.css`.

**Case Project 1-3**

Create a document with an `<h1>` element containing the text “Forestville Funding” and an `<h2>` element containing the text “Auto Loan Rates.” Use the strict DTD and link the document to the `js_styles.css` style sheet, located in your Cases folder for Chapter 1. Add a script section containing a CDATA section to the document body. Within the CDATA section, use the `document.write()` statement to print the following auto loan rate information: 5.85% (24 Month Terms), 6.25% (36 Month Terms), and 7.65% (48 Month Terms). Print each line as an item in an unordered list by using the `<ul>` and `<li>` elements. Use another `document.write()` statement to print a `<p>` element after the unordered list that contains the text “Minimum down payment: 10%”. Add JavaScript comments with your name, today’s date, and Case Project 1-3. Include code to hide the JavaScript code from incompatible browsers. Save the document as `AutoRates.html`.
Case Project 1-4

Create a document with three script sections: one in the document head and two in the document body. Use the strict DTD and link the document to the js_styles.css style sheet, located in your Cases folder for Chapter 1. Be sure to include CDATA sections within the script sections. In the script section in the document head, include a `document.write()` statement that prints a line that reads "<h1>Don's Jungle Tours</h1>". Be sure to include the heading element. Add <h2>Adventure</h2> above the first script section and <h2>Excellence</h2> above the second script section. Add JavaScript comments with your name, today’s date, and Case Project 1-4. In the first script section in the document body, use five `document.write()` statements to print an unordered list containing the following three lines: “Ecotourism is our specialty”, “Get up close and personal with nature”, and “Destinations include Africa, South America, and Asia”. The second script section in the document body should call a JavaScript source file that prints the following two lines as list items: “Best quality and price” and “Authentic in-country experience”. Save the Web page document as JungleTours.html and the JavaScript source file as travel.js.

Case Project 1-5

The Cases folder for Chapter 1 contains a Web page named HighestWaterfalls.html that uses `document.write()` statements to print a table containing the names, locations, and height of the 10 highest waterfalls in the world. Both the XHTML elements and the JavaScript statements contain errors that prevent the Web page from validating against the strict DTD and the `document.write()` statements from functioning. Identify and fix the problems in the file. Be sure that the document validates against the strict DTD, prints the names, locations, and heights for all 10 waterfalls, and does not generate any JavaScript errors.