to the end of the filename. To save a program file using Notepad, follow these steps:

1. **Choose File ➪ Save As from the menu bar at the top of the editor.**

   The Save As dialog box, shown in Figure 3-2, appears.

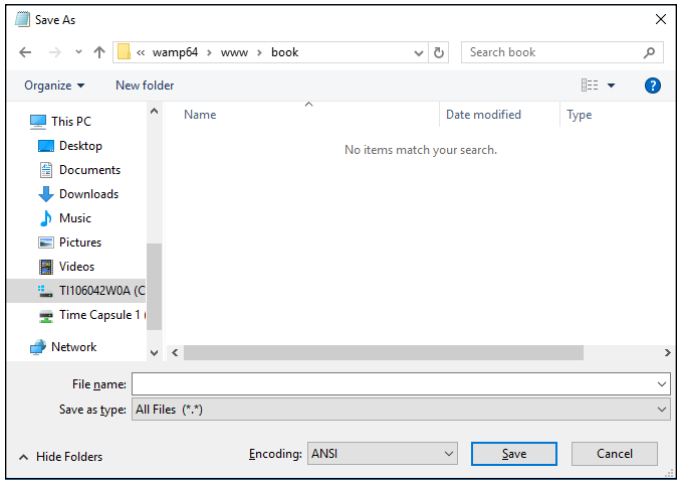

FIGURE 3-2: The Microsoft Notepad Save As dialog box.

2. **In the drop-down list at the top of the Save As dialog box, navigate to the folder where you want to save the program file.**
3. **From the Save As Type text box near the bottom of the Save As dialog box, select All Files (*.*).**

   This prevents Notepad from appending the `.txt` file extension to your filename.
4. **In the File Name field, enter the filename for your program file, including the file extension you want to use.**
5. **Click Save to save the program file.**

   Your program file is properly saved in the correct format, with the correct filename, in the correct location.

SEEING FILE EXTENSIONS

In Microsoft Windows you use File Explorer to navigate the storage devices on your system to open files. Unfortunately, the default setup in File Explorer is to hide the file extension part of the filename (the part after the period) so that it doesn't confuse novice computer users.

That can have the opposite effect for programmers, adding confusion when you're trying to look for a specific file. You may use the same filename for multiple files with different extensions. Fortunately, you can easily change this default setting in Windows. Just follow these steps:

1. **In Windows 8 or 10, open Settings. In Windows 7, open the Control Panel.**
2. **In Windows 8 or 10, type** File Explorer Options **in the search bar and press Enter.**
3. **Click the icon for the File Explorer Options tool that appears in the search results.**
4. **In Windows 7, click the File Explorer Options icon in the Control Panel.**

   You may have to go to the Advanced view to see it.

   After you open File Explorer Options, the dialog box should look like Figure 3-3.

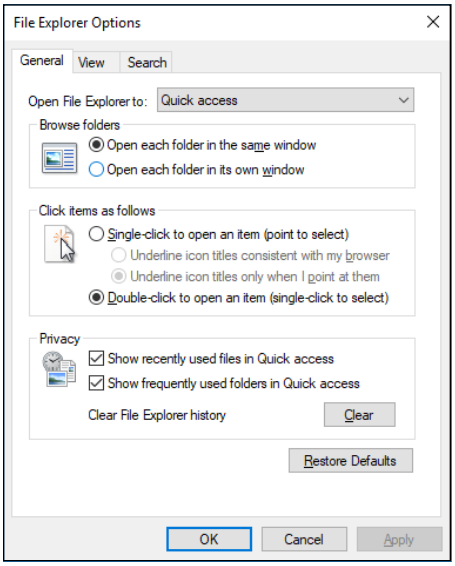

FIGURE 3-3: The File Explorer Options dialog box in Windows.

5. **Click the View tab.**
6. **Remove the check mark from the Hide Extensions for Known File Types check box, as shown in Figure 3-4.**
7. **Click OK.**

   Now you'll be able to see the full filename, including the extension, when you look for your programs using File Explorer.

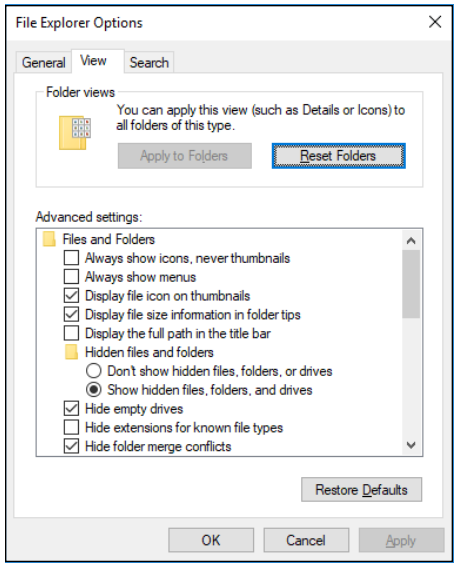

FIGURE 3-4: Removing the Hide Extensions for Known File Types check mark.

SETTING THE DEFAULT APPLICATION

Now that you can see the full filename of your program files in File Explorer, there's just one more hurdle to cross. If you want to open your program files using Notepad by default, you'll need to tell File Explorer to do that. Follow these steps:

1. **Navigate to the program file, and right-click the filename.**
2. **In the menu that appears, select Open.**

   The Open dialog box appears.
3. **Select Notepad from the list of programs, and then select the check box to always open files of this type using the program.**

   Now you'll be able to double-click your program files in File Explorer to automatically open them in Notepad.

If you're running macOS

If you're running macOS (or one of the earlier Mac OS X versions), the text editor that comes standard is called TextEdit. The TextEdit application actually provides quite a lot of features for a standard text editor — it recognizes and allows you to edit a few different types of text files, including rich text files (`.rtf`) and HTML files.

The drawback to TextEdit is that sometimes it can be *too* smart. Trying to save and edit an HTML file in TextEdit can be more complicated than it should be. By default, TextEdit will try to display the HTML tags as their graphical equivalents in the editor window, as shown in Figure 3-5.

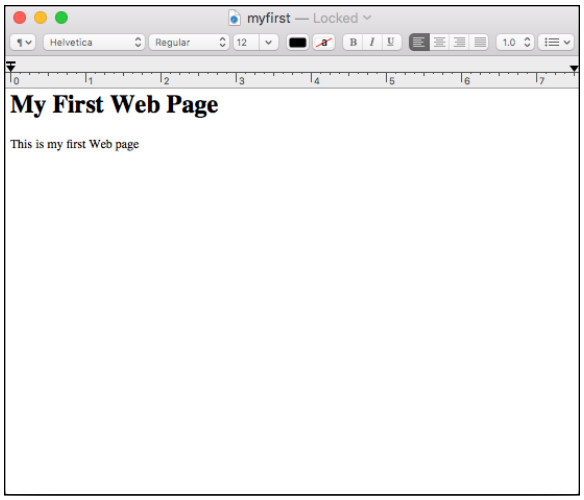

FIGURE 3-5: Using the default TextEdit settings to edit an HTML file.

As you can see in Figure 3-5, TextEdit actually shows the text as the HTML tags format it instead of the actual HTML code. This won't work for editing an HTML file, because you need to see the code text instead of what the code generates. There's an easy way to fix that — just follow these steps:

1. **Choose TextEdit ➪ Preferences.**

   The Preferences dialog box, shown in Figure 3-6, appears.

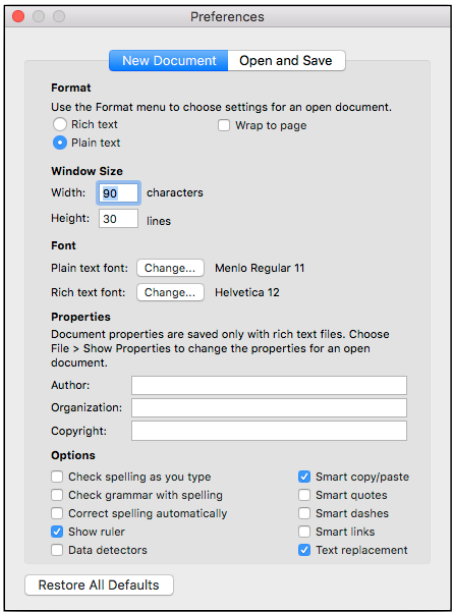

FIGURE 3-6: The Preferences dialog box in TextEdit.

2. **On the New Document tab, in the Format section, select the Plain Text radio button.**
3. **In the Options section, remove the check mark from the following check boxes:**
   - Correct Spelling Automatically
   - Smart Quotes
   - Smart Dashes
   - Smart Links
4. **Click the Open and Save tab (see Figure 3-7).**

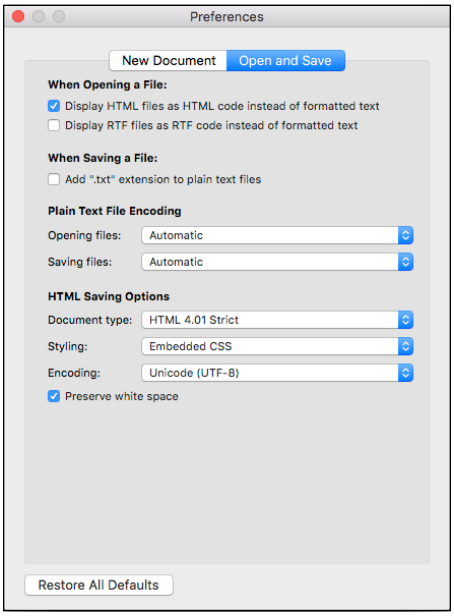

FIGURE 3-7: The Open and Save tab of the Preferences dialog box.

5. **In the When Opening a File section, check the Display HTML Files as HTML Code Instead of Formatted Text check box.**
6. **In the When Saving a File section, remove the check mark from the Add ".txt" File Extension to Plain Text Files check box.**
7. **Close the Preferences dialog box to save the settings.**

Now you're all set to start editing your program code using TextEdit!

If you're running Linux

The Linux environment was made by programmers, for programmers. Because of that, even the simple text editors installed by default in Linux distributions provide some basic features that come in handy when coding.

Which text editor comes with your Linux distribution usually depends on the desktop environment. Linux supports many different graphical desktop environments, but the two most common are GNOME and KDE. This section walks through the default text editors found in each.

THE GNOME EDITOR

If you're working in a GNOME desktop environment, the default text editor is gedit, shown in Figure 3-8.

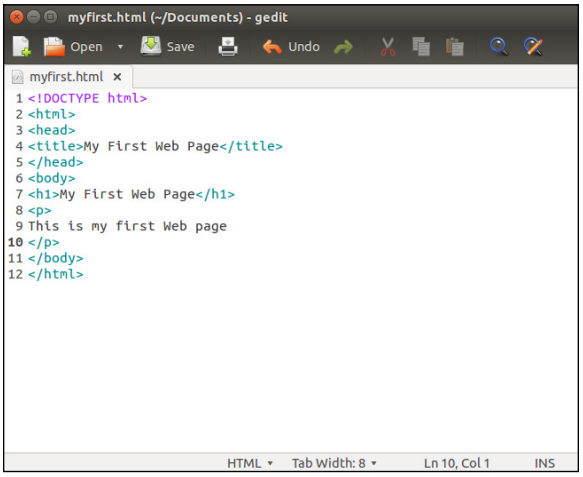

FIGURE 3-8: The gedit editor used in Linux GNOME desktops.

The gedit editor automatically saves program files as plain text format and doesn't try to add a `.txt` file extension to filenames. There's nothing special you need to do to dive into coding your programs using gedit. Plus, it has some advanced features specifically for programming that you would find in program editors (see the "Program editors" section later in this chapter).

THE KDE EDITOR

The default text editor used in the KDE graphical desktop environment is Kate, shown in Figure 3-9.

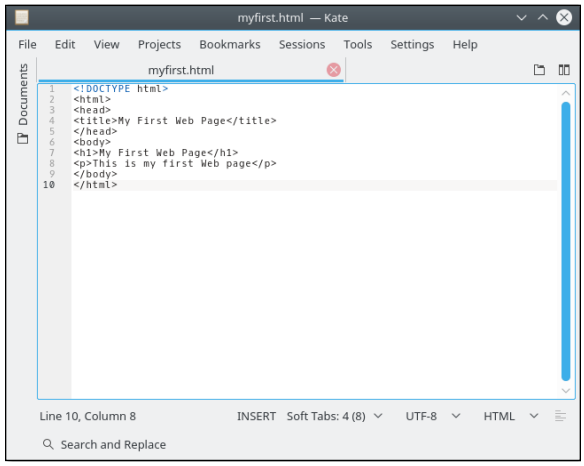

FIGURE 3-9: The Kate editor used in Linux KDE desktops.

Just like gedit, the Kate editor contains lots of programmer-friendly features right out of the box. Again, no special configuration is required before you can start editing your program code in Kate.

Program editors

The next step up from standard text editors is a family of tools called *program editors.* A program editor works just like a text editor, but it has a few additional built-in features that come in handy for programming. Here are some of the features that you'll find in program editors:

- **Line numbering:** Providing the line numbers off to the side of the window is a lifesaver when coding. When an error message tells you there's a problem on line 1935, not having to count every line to get there helps!
- **Syntax highlighting:** With syntax highlighting, the editor uses different colors for different parts of the program. Program keywords are displayed using different colors to help make them stand out from data in the code file.
- **Syntax error marking:** Text that appears to be used as a keyword but that isn't found in the code statement dictionary is marked as an error. This feature can be a time-saver by helping you catch simple typos in your program code.

There are lots of commercial program editors, but some of the best program editors are actually free. This section discusses some of the better free ones available for HTML, CSS, JavaScript, and PHP coding.

Notepad++

If you're running Microsoft Windows, the Notepad++ tool is a great place to start. As its name suggests, it's like Notepad, but better. You can download Notepad++ from `www.notepad-plus-plus.org`. The main editing window is shown in Figure 3-10.

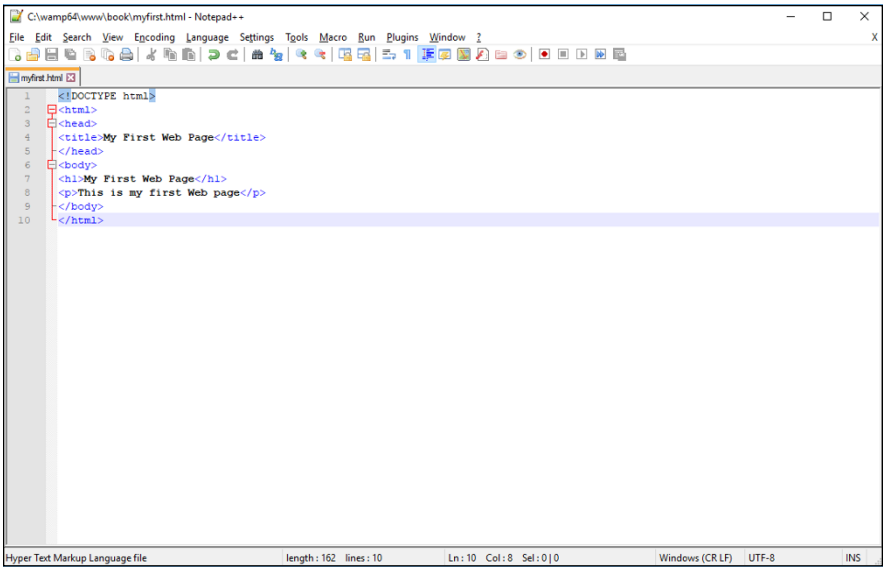

FIGURE 3-10: Notepad++.

The main interface for Notepad++ looks similar to Notepad, so there's nothing different to get used to. By default, it shows line numbers along the left margin, as well as the type of file and the column location of the cursor at the bottom.

Notepad++ recognizes the syntax for many different types of programming languages, including HTML, CSS, JavaScript, and PHP. It highlights the keywords and will even match up opening and closing block statements. If you miss a closing block, Notepad++ will point that out.

Scintilla and SciTE

The Scintilla library (`www.scintilla.org`) is an open-source project to provide a programming text editor engine for use in any type of environment. Developers can embed the Scintilla editor into any type of application free of charge.

The SciTE package is a desktop text editor tool that implements the Scintilla library. The SciTE package is available for Windows, macOS, and Linux platforms. You can download it from the Scintilla website for the Windows and Linux

platforms, and it's available in the Apple Store for the macOS platform. Figure 3-11 shows the basic SciTE editor window in action.

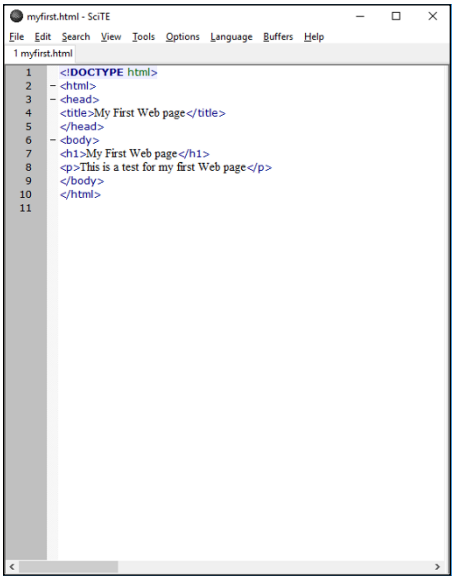

FIGURE 3-11: SciTE.

SciTE provides all the program editing features mentioned earlier. It recognizes the syntax of many different programming languages and can help you organize your code by marking and collapsing code sections (this comes in handy if you write long if-then statement sections).

jEdit

The jEdit program editor (`www.jedit.org`) is a little bit different from the other packages. It's written in Java code, so you can run it in any platform that supports Java. That means you can use the exact same editor interface in Windows, macOS, or Linux! jEdit supports all the common features you'd expect from a program editor. Figure 3-12 shows the basic jEdit editor window.

WARNING

Because jEdit is a Java application, your desktop platform must have either a Java Runtime Environment (JRE) or Java Development Kit (JDK) package installed in order for it to work. You can download and install one from Oracle at `www.oracle.com/technetwork/java/javase/downloads`. Also, because jEdit runs as a Java application, you may find it slower than some of the native desktop packages such as Notepad++ or SciTE.

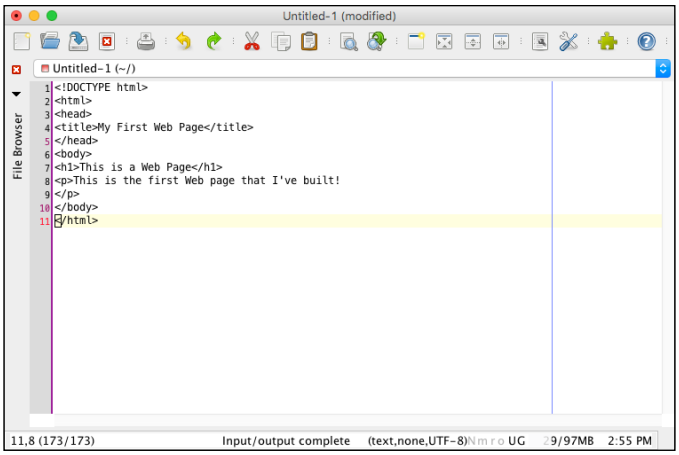

FIGURE 3-12: jEdit.

Integrated development environments

Moving up the ladder of tools, the laser-guided miter tool for program development is the *integrated development environment* (IDE). IDE packages provide everything you could possibly need to develop any size of web application.

Here are some of the advanced features IDE packages provide:

- **Code completion:** Start typing a code statement, and the package will provide a pop-up list of statements that match what you're typing. It also shows what parameters are required and optional for the statement.
- **Code formatting:** The IDE automatically indents code blocks to help make your code more readable.
- **Program execution:** You can run your code directly from the editor window without having to jump out to a web browser.
- **Debugging:** You can step through the program code line by line, watch how variables are set, and see whether any error messages are generated.
- **Project and file management:** Most IDE packages allow you to group your application files into projects. This allows you to open a project and see just the files associated with that application. Some will even upload the project files to your web-hosting site for you, similar to what the graphical desktop tools do.

Using an IDE tool is not for the faint of heart. Because of all the fancy features, learning how to use the IDE interface can be almost as complicated as learning the programming language!

There are both commercial and open-source IDE packages available for the PHP environment. To give you a general idea of how IDE packages operate, this section walks through two of the more popular ones: Netbeans and Eclipse.

Netbeans

The Netbeans IDE package was originally developed by Sun Microsystems and released as an open-source IDE for its Java programming language environment (thus the "beans" part of the name). When Oracle acquired Sun, it maintained support for Netbeans, and continued development of it with updated releases.

The Netbeans IDE now contains support for several different programming languages besides Java by using additional plug-in modules. As you can guess, the reason I'm mentioning it here is because there's a plug-in module for PHP.

You can download the Netbeans editor with the PHP module already installed, making it easy to install. Just go to `www.netbeans.org/downloads` and click the Download button under the PHP category.

When you start Netbeans, it will prompt you to start a new project, as shown in Figure 3-13.

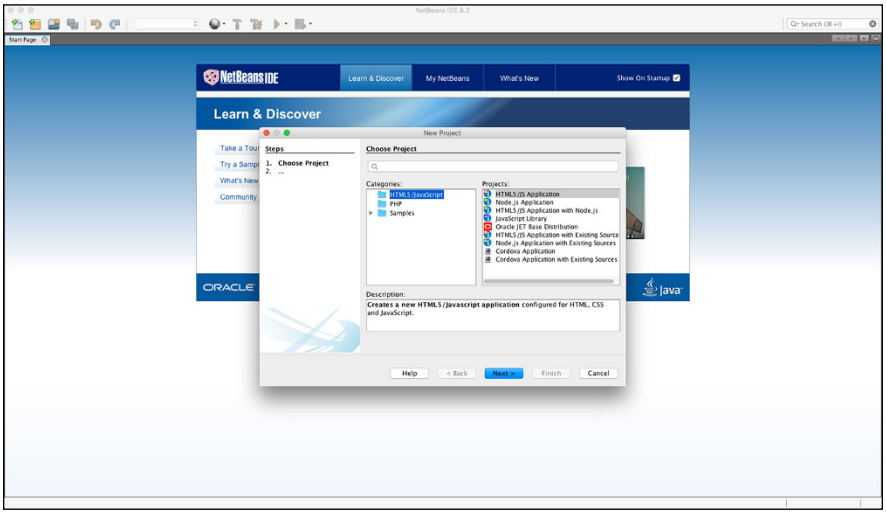

FIGURE 3-13: The Netbeans project dialog box.

Netbeans contains project templates for HTML and JavaScript applications, as well as PHP applications. When you start a new PHP project, Netbeans automatically creates an `index.php` file as the main program file for the project. It even builds a rough template for your code. As you would expect from an IDE, when you

start typing a PHP function name, Netbeans opens a pop-up box that shows all the PHP functions that match what you're typing, as shown in Figure 3-14.

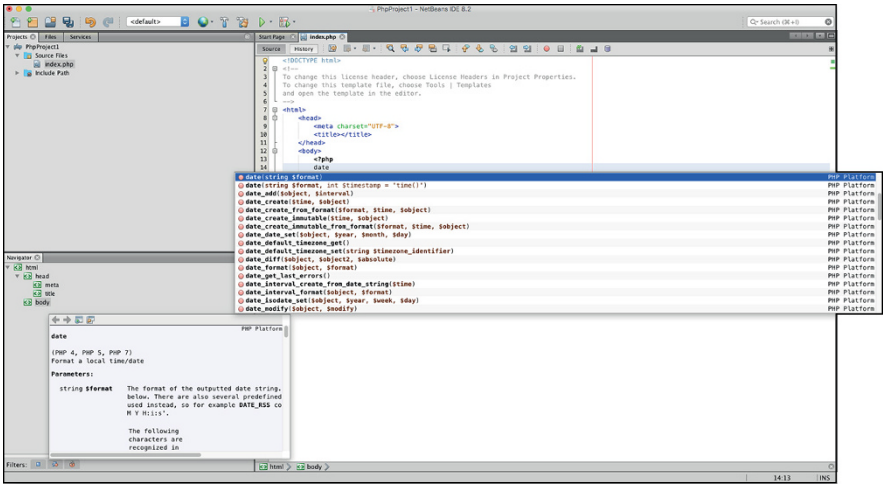

FIGURE 3-14: The Netbeans code completion dialog box.

Not only does it show the code completion list, as you can see in Figure 3-14, but it also shows you the PHP manual definition of the function! This is certainly a handy tool to have available if you plan on doing any large-scale PHP development.

Eclipse

The other big name in PHP IDE packages is the Eclipse PHP Development Tool (usually just called Eclipse PDT). Eclipse was also originally designed as a Java application IDE. Many open-source proponents didn't trust Sun Microsystems maintaining the only IDE for Java, so they set out to develop their own. (The story goes that there was no intentional wordplay on the name Eclipse versus Sun Microsystems. If you can believe that, I may have a bridge to sell you.)

Just like the Netbeans IDE, Eclipse evolved from a Java-only IDE to support many different programming languages via the use of plug-in modules. You can download the Eclipse PDT as an all-in-one package at `www.eclipse.org/pdt`.

WARNING

Just like the jEdit editor, Eclipse PDT is written as a Java application and requires that you have a JRE or JDK installed on your workstation (see "jEdit," earlier in this chapter).

When you start Eclipse, a menu system appears, as shown in Figure 3-15.

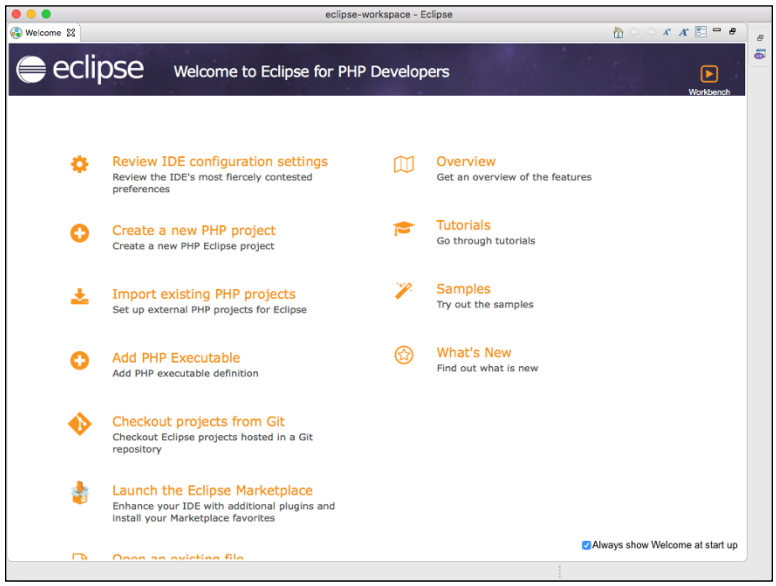

FIGURE 3-15: The Eclipse start menu.

This allows you to easily change the IDE configuration, start a new project, or open an existing project. Eclipse has all the same features that Netbeans offers. Plus, it has one additional feature: Eclipse PDT contains the advanced PHP Debugger tool developed by Zend, the company that sponsors PHP. The Debugger tool can help point out errors in your PHP code immediately as you type, or it can debug your code as you run it in the Eclipse editor window. Figure 3-16 demonstrates Eclipse pointing out a PHP coding error I made in my code.

Having an advanced PHP debugger at your fingertips can be a great time-saver when you're developing large applications!

Browser debuggers

Before I finish this chapter, I want to mention one more tool that you have available when trying to troubleshoot web application issues. Most browsers today have a code-debugging feature either built in or easily installable. The browser debuggers can help you troubleshoot HTML, CSS, and JavaScript issues in the web page you send to the client. Figure 3-17 shows the debugging console in the Microsoft Edge web browser after you press F12 to activate it.

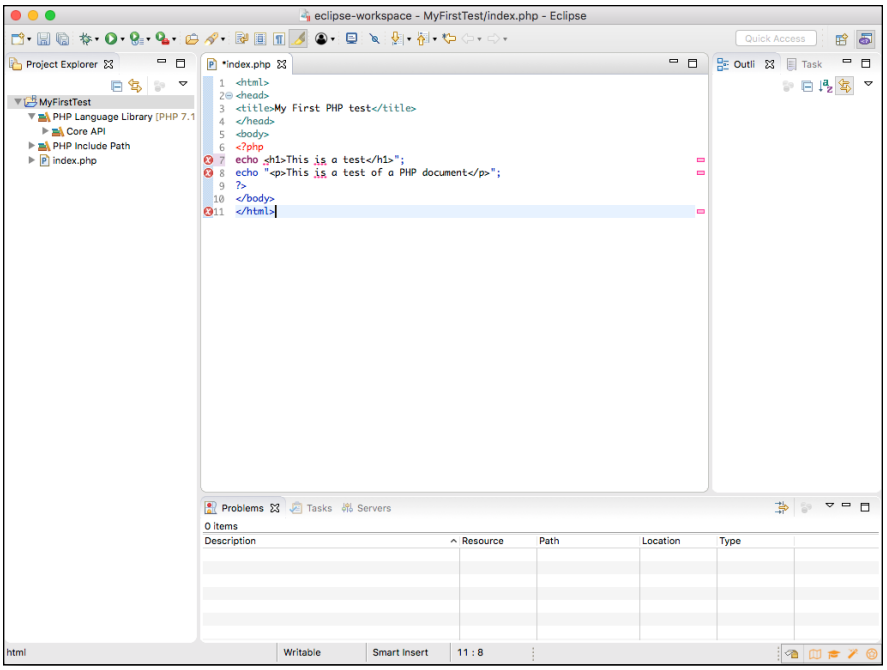

FIGURE 3-16: The PHP debugger in action in Eclipse.

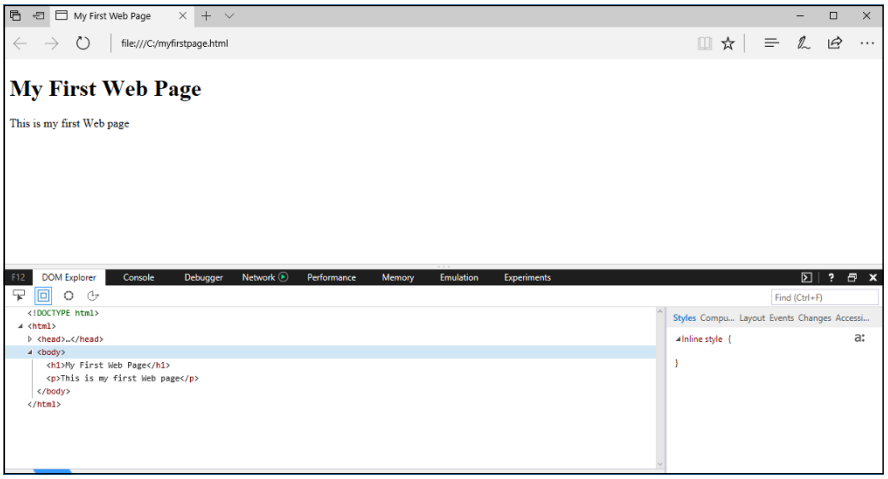

FIGURE 3-17: The Microsoft Edge web browser debugging a web page.

Browser debuggers can show you exactly where something has gone wrong in the HTML or CSS code. They're also invaluable when working with JavaScript applications.

When you're developing web applications, it's crucial that you test, do some more testing, and then test again. Testing your application in every possible way your website visitors will use it is the only way to know just what to expect.

Things are getting better, but different browsers still may handle HTML, CSS, and even JavaScript code differently. Nowhere is this more evident than when errors occur.

When an error occurs in HTML or CSS code, the browser doesn't display any type of error message. Instead, it tries to fix the problem on its own so it can display the web page. Unfortunately, not all browsers fix code the same way. If you run into a situation where your web page looks different on two different browsers, most likely you have some type of HTML or CSS code issue that the browsers are interpreting differently.

2

HTML5 and CSS3

Contents at a Glance

IN THIS CHAPTER

- Looking at the HTML5 document structure
- Identifying the basic HTML5 elements
- Formatting text
- Using special characters
- Creating lists
- Working with tables

Chapter 1

The Basics of HTML5

The core of your web application is the HTML5 code you create to present the content to your site visitors. You need an understanding of how HTML5 works and how to use it to best present your information. This chapter describes the basics of HTML5 and demonstrates how to use it to create web pages.

Diving into Document Structure

The HTML5 standard defines a specific structure that you must follow when defining your web pages so that they appear the same way in all browsers. This structure includes not only the markups that you use to tell browsers how to display your web page content, but also some overhead information you need to provide to the browser. This section explains the overall structure of an HTML5 program, and tells you what you need to include to ensure your clients' browsers know how to work with your web pages correctly.

Elements, tags, and attributes

An HTML5 document consists of one or more elements. An *element* is any object contained within your web page. That can be headings, paragraphs of text, form

fields, or even multimedia clips. Your browser works with each element individually, positioning it in the browser window and styling it as directed.

You define elements in your web page by using tags. A *tag* identifies the type of element so the browser knows just how to handle the content it contains. The HTML5 specification defines two types of elements:

- **Two-sided elements:** Two-sided elements are the more common type of element. A two-sided element contains two parts: an *opening tag* and a *closing tag*. The syntax for a two-sided element looks like this:

```
<element>content</element>
```

  The first element tag is the opening tag. It contains the element name, surrounded by the less-than symbol (<) and greater-than symbol (>), and defines the start of the element definition.

  The second tag is the closing tag; it defines the end of the element definition. It points to the same element name, but the name is preceded by a forward slash (/). The browser should handle any content between the two tags as part of the element content. For example, the HTML5 `h1` element defines a heading like this:

```
<h1>This is a heading</h1>
```

  The element instructs the browser to display the text *This is a heading* using the font and size appropriate for a heading on the web page. It's up to the browser to determine just how to do that.

- **One-sided elements:** One-sided elements don't contain any content and usually define some type of directive for the browser to take in the web page. For example, the line break element instructs the browser to start a new line in the web page:

```
<br>
```

  Because there's no content, there's no need for a closing tag.


TIP

  The older XHTML standard requires that one-sided tags include a closing forward slash character at the end of the tag, such as `<br/>`. This isn't required by HTML5, but it's supported for backward compatibility. It's very common to still see that format used in HTML5 code.

Besides the basic element definition, many elements also allow you to define attributes to apply to the element. *Attributes* provide further instructions to the browser on how to handle the content contained within the element. When you define an attribute for an element, you must also assign it a *value*.

You include attributes and their values inside the opening tag of the element, like this:

```
<element attribute="value">content</element>
```

You can define more than one attribute/value pair for the element. Just separate them using a space in the opening tag:

```
<element attribute1="value1" attribute2="value2">
```

Attributes are commonly used to apply inline styles to elements:

```
<h1 style="color: red">Warning!!</h1>
```

The `style` attribute shown here defines additional styles the browser should apply to the content inside the element. In this example, the browser will change the font color of the text to red.

Document type

Every web page must follow an HTML or XHTML document standard so the browser can parse it correctly. The very first element in the web page code is the markup language standard your document follows. This element, called the *document type,* is crucial, because the browser has to know what standard to follow when parsing the code in your web page.

You define the document type using the `<!DOCTYPE>` tag. It contains one or more attributes that define the markup language standard. Prior versions of HTML used a very complicated format for the document type definition, pointing the browser to a web page on the Internet that contained the standard definition.

Fortunately, the HTML5 standard reduced that complexity. To define an HTML5 document, you just need to include the following line:

```
<!DOCTYPE html>
```

When the browser sees this line at the start of your web page code, it knows to parse the elements using the HTML5 standard.

WARNING

If you omit the `<!DOCTYPE>` tag, the browser will still attempt to parse and process the markup code. However, because the browser won't know exactly which standard to follow, it follows a practice known as *quirks mode.* In quirks mode, the browser follows the original version of the HTML standard, so newer elements won't be rendered correctly.

Page definition

To create an HTML5 web page, you just define the different elements that appear on the page. The elements fit together as part of a hierarchy of elements. Some elements define the different sections of the web page, and other elements contained within those sections define content.

The *html element* is at the top of the hierarchy. It defines the start of the entire web page. All the other elements contained within the web page should appear between the `<html>` opening and `</html>` closing tags:

```
<!DOCTYPE html>
<html>
web page content
</html>
```

Most Web pages define at least two second-level elements, the head and the body:

```
<html>
<head>
head content
</head>
<body>
body content
</body>
</html>
```

The *head element* contains information about your web page for the browser. Content contained within the head element doesn't appear on the web page, but it directs things behind the scenes, such as any files the browser needs to load in order to properly display the web page or any programs the browser needs to run when it loads the web page.

One element that's commonly found in the head element content is the title, which defines the title of your web page:

```
<head>
<title>My First Web Page</title>
</head>
```

The web page title isn't part of the actual web page, but it usually appears in the browser's title bar at the top of the browser window or in the window tab if the browser supports tabbed browsing.

The *body element* contains the elements that appear in the web page. This is where you define the content that you want your site visitors to see. The body element

should always appear after the head element in the page definition. It's also important to close the body element before closing out the html element.

Follow these steps to create and test your first web page:

1. **Open the editor, program editor, or integrated development environment (IDE) package of your choice.**

   See Book 1, Chapter 3, for ideas on which tool to use.

2. **Enter the following code into the editor window:**

```
<!DOCTYPE html>
<html>
<head>
<title>My First Web Page</title>
</head>
<body>
This is text inside the web page.
</body>
</html>
```

3. **Save the code to the `DocumentRoot` folder of your web server, naming it `mytest.html`.**

   If you're using the XAMPP server in Windows, the folder is `c:\xampp\htdocs`. For macOS, it's `/Applications/xampp/htdocs`.

4. **Start the XAMPP servers.**

5. **Open the browser of your choice, and enter the following URL:**

```
http://localhost:8080/mytest.html
```

   Note that you may need to change the 8080 port number specified in the URL to match your XAMPP Apache server set up (see Book 1, Chapter 2). Figure 1-1 shows the web page that this code produces.

The head element defines the web page title, which as shown in Figure 1-1, appears in the web browser title bar. The body element contains a single line of text, which the browser renders inside the browser window area.


TIP

You may notice that other than the special `<!DOCTYPE>` tag, all the other HTML tags I used are in lowercase. HTML5 ignores the case of element tags, so you can use uppercase, lowercase, or any combination of the two for the element names in the tags. The older XHTML standard requires all lowercase tags, so many web developers have gotten into the habit of using lowercase for tags, and more often than not, you'll still see HTML5 code use all lowercase tag names.

FIGURE 1-1: The output for the sample web page.

Page sections

Web pages these days aren't just long pages of content. They contain some type of formatting that lays out the content in different sections, similar to how a newspaper presents articles. In a newspaper, usually there are two or more columns of content, with each column containing one or more separate articles.

In the old days, trying to create this type of layout using HTML was somewhat of a challenge. Fortunately, the HTML5 standard defines some basic elements that make it easier to break up our web pages into sections. Table 1-1 lists the HTML5 elements that you use to define sections of your web page.

TABLE 1-1 HTML5 Section Elements

Element	Description
`article`	A subsection of text contained within a section
`aside`	Content related to the main article, but placed alongside to provide additional information
`div`	A grouping of similarly styled content within an article
`footer`	Content that appears at the bottom of the web page
`header`	Content that appears at the top of the web page
`nav`	A navigation area allowing site visitors to easily find other pages or related websites
`section`	A top-level grouping of articles

REMEMBER

Although HTML5 defines the sections, it doesn't define how the browser should place them in the web page. That part is left up to CSS styling, which I talk about in Chapter 2 of this minibook.

When you combine the HTML5 section elements with the appropriate CSS3 styling, you can create just about any look and feel for your web pages that you want. Although there's no one standard, there are some basic rules that you can follow when positioning sections in the web page. Figure 1-2 shows one common layout that I'm sure you've seen used in many websites.

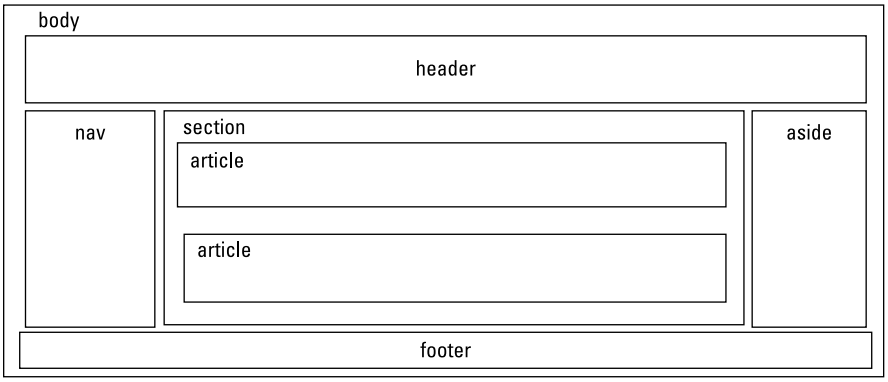

FIGURE 1-2: A basic web page layout using HTML5 section elements.

Just about every web page has a heading section at the top of the page that identifies it to site visitors. After that, a middle section is divided into three separate areas. On the left side is often a navigation section, providing links to other pages in the website. On the right side is often additional information or, in some cases, advertisements. In the middle of the middle section is the meat of the content you're presenting to your site visitors. Finally, at the bottom of the web page is a footer, often identifying the copyright information, as well as some basic contact information for the company.

The *div element* is a holdout from previous versions of HTML. If you need to work with older versions of HTML, instead of using the named section elements, you need to use the `<div>` tag, along with the `id` attribute to define a specific name for the section:

```
<div id="header">
content for the heading
</div>
```

The CSS styles refer to the `id` attribute value to define the styles and positioning required for the section. You can still use this method in HTML5. Designers often use the div element to define subsections within articles that need special styling.

A WORD ABOUT WHITE SPACE

Quite possibly the most confusing feature in HTML is how it uses white space. The term *white space* refers to spaces, tabs, consecutive spaces, and line breaks within the HTML code.

By default, when a browser parses the HTML code, it ignores any white space between elements. So, these three formats all produce the same results:

```
<title>
My First Web Page
</title>

<title>My First Web Page
</title>

<title>My First Web Page</title>
```

It's completely up to you which format to use for your programs, but I recommend choosing a format and sticking to it. That'll make reading your code down the road easier, for you or anyone else.

COMMENTING YOUR CODE

Every programming language allows you to embed comments inside the code to help with documenting what's going on. HTML is no different. HTML allows you to insert text inside the HTML document that will be ignored by the browser as it parses the text.

To start a comment section in HTML, you use the following symbol:

```
<!--
```

You can then enter as little or as much text as you need to properly document what's going on in your code. When the comment text is complete, you have to close the comment section using the following symbol:

```
-->
```

You can place anything between the opening and closing comment tags, including HTML code, and the browser will ignore it. However, be careful what you say in your comments, because they can be read by anyone who downloads your web page!

Now that you know how to define different sections of the web page, the next section discusses how to add content to them.

Looking at the Basic HTML5 Elements

After you define one or more sections in your web page, you're ready to start defining content. Adding content to a web page is sort of like working on a car assembly line. You define each piece of the web page separately, element by element. It's up to the browser to assemble the pieces to create the finished web page.

This section covers the main elements that you'll use to define content in your web page.

Headings

Normally, each new section of content in a web page will use some type of heading to make it stand out. Research shows that the first thing site visitors usually do when visiting a web page is to scan the main headings on the page. If you can't attract their attention with your section headings, you may quickly lose them.

HTML5 uses the *h element* to define text for a heading. It defines six different levels of headings. Each heading level has a separate tag:

```
<h1>A level 1 heading</h1>
<h2>A level 2 heading</h2>
<h3>A level 3 heading</h3>
<h4>A level 4 heading</h4>
<h5>A level 5 heading</h5>
<h6>A level 6 heading</h6>
```

Although there are six levels of headings in the HTML5 standard, most sites don't use more than two or three.

The client browser determines the font, style, and size of the text it uses for each heading level. Figure 1-3 shows how the Chrome web browser interprets the six levels of headings.

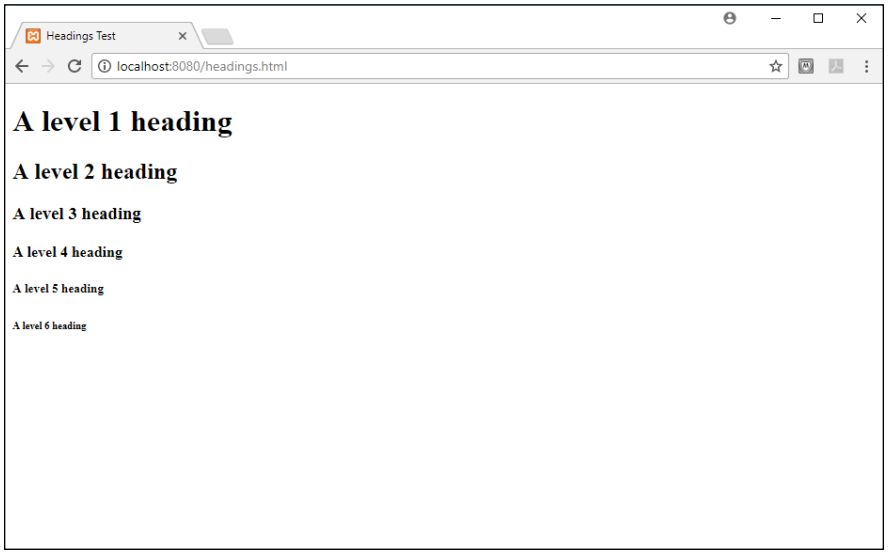

FIGURE 1-3: Displaying all six heading levels in the Chrome web browser.

The browser displays each heading level with a decreasing font size. By the time you get to the sixth heading level, it's pretty hard to tell the difference between the heading and normal text on the web page!

Text groupings

There are several HTML5 elements that allow you to group text together into what are called *block-level elements*. The browser treats all of the content defined within the opening and closing tags of a block-level element as a single group. This allows you to use CSS to style or position the entire block of content as one piece, instead of having to style or position each element individually.

You can group headings together using a new feature in the HTML5 standard called a *heading group*, using the *hgroup* element:

```
<hgroup>
<h1>This is the main heading.</h1>
<h2>This is the subheading.</h2>
</hgroup>
```

The heading group doesn't change the h1 or h2 elements, but it provides a way for the browser to interpret the two headings as a single element for styling and positioning. This allows you to use CSS styles to format them as a single block so they blend together like a main heading and a subheading.

A web page consisting of sentences just strung together is boring to read and won't attract very many site visitors (or may just put them to sleep). In print, we

group sentences of common thoughts together into paragraphs. You do the same thing in your web page content by using the *p element:*

```
<p>This is one paragraph of text. The paragraph contains two sentences of
   content.</p>
```

Notice that the p element uses an opening tag (`<p>`) and a closing tag (`</p>`) to mark the beginning and end of the grouped text. The browser treats all the text inside the p element as a single element. When you group the content together, you can apply styles and positioning to the entire block.

Be careful with the p element, though. The rules of white space that apply to HTML tags also apply to text inside the p element. The browser won't honor line breaks, tabs, or multiple spaces. So, if you have code like this:

```
<p>
This is one        line.
This is     another line.
</p>
```

It will appear in the web page like this:

```
This is one line. This is another line.
```

All the extra spaces and the line break are removed from the content. Also, notice that the web browser adds a space between the two sentences.

If you want to preserve the formatting of the text in the web page, use the *pre element.* The pre element allows you to group preformatted text. The idea behind preformatted text is that it appears in the web page exactly as you enter it in the code file:

```
<pre>
This is one        line.
This is     another line.
</pre>
```

The browser will display the text in the web page exactly as it appears in the HTML5 code.

Yet another method of grouping text is the *blockquote element.* The blockquote element is often used to quote references within a paragraph. The browser will

indent the text contained within the blockquote separate from the normal paragraph text:

```
<p>The only poem that I learned as a child was:</p>
<blockquote>Roses are red, violets are blue. A face like yours, belongs in the
   zoo.</blockquote>
<p>But that's probably not considered classic poetry.</p>
```

This feature helps you embed any type of text within content, not just quotes.

Breaks

Because HTML doesn't recognize the newline character in text, there's a way to tell the browser to start a new line in the web page when you need it. The single-sided *br element* forces a new line in the output:

```
<p>
This is one line.
<br>
This is a second line.
</p>
```

Now the output in the web page will appear as:

```
This is one line.
This is a second line.
```

Another handy break element is the *hr element.* It displays a horizontal line across the width of the web page section.

```
<h1>Section 1</h1>
<p>This is the content of section 1.</p>
<hr>
<h1>Section 2</h2>
<p>This is the content of section 2.</p>
```

The horizontal line spans the entire width of the web page block that contains it, as shown in Figure 1-4.

Sometimes that's a bit awkward, but you can control the width of the horizontal line a bit by enclosing it in a section and adding some CSS styling.