IN THIS CHAPTER

- Finding out a bit about what you're getting yourself into
- Befriending HTML
- Introducing yourself to CSS

Chapter 1
Getting to Know HTML and CSS

This book is a sort of manual for using HTML and CSS. However, and this is particularly true if you're just getting started with coding web pages, if there's any part of the book that fits the old RTFM (read the freakin' manual) credo, it's this chapter. *Everything* you learn in this chapter acts as a kind of home base for the explorations that come later.

In this chapter, you learn the basic concepts behind HTML and CSS, get a better understanding of how they work, and get started exploring these powerful technologies.

Getting a Bird's-Eye View of the Process

You can add special codes inside a text file to specify how you want your web page to look. For example, maybe you want a particular collection of words or phrases to appear as a bulleted list. When the web browser comes to that part of the file, it dutifully renders those items as a list, bullets and all. The person browsing your page doesn't get the "render these items as a bulleted list" code; they just get the bulleted list. The web browser performs

these and many other transformations behind the scenes. As long as you have the right HTML and CSS markings in the right places, the browser will render your page the way you want.

Launching a new text file

Your first step whenever you want to create a web page is to start a new text file. To do that, not surprisingly, you need to fire up your favorite text editor:

- **Notepad (Windows):** In Windows 11, choose Start ➪ All Apps ➪ Notepad. Notepad displays a brand-new text file automatically when you start the program. You can also fire up a new document by choosing File ➪ New.
- **TextEdit (Mac):** Click Search (the magnifying glass) in the menu bar, start typing **textedit**, and then click TextEdit as soon as it shows up in the search results. In the dialog box that appears, click New Document. You can launch a new text file any time you need one by choosing File ➪ New.
- **Something else:** If you have another text editor, launch it the way you normally do and create a new file.

Saving HTML files

When it's time to save your work, here are a few notes to consider:

- **Use the right file extension.** For garden-variety web pages, your file names must end with either the `.htm` or the `.html` file extension (for example, `mypage.html`).
- **Use lowercase filenames without spaces.** For best results, always enter your filenames using only lowercase letters and don't use spaces in your filenames. If you want to separate words in file and directory names, use an underscore (_) or a hyphen (–).
- **Use the right file type.** While in the Save As dialog box, you need to select the correct file type for your HTML file. How you do this depends on what program you're using. In most programs (including TextEdit), you use the File Format (or Save As Type) list to select Web Page (`.html`) or something similar. If you're using Notepad, use the Save As Type list to select All Files (*.*). This ensures that Notepad uses your `.htm` or `.html` extension (and not its normal `.txt` extension).

Edit. Save. Reload. Repeat.

Assuming that you've previously saved your HTML file as I describe in the previous section, your first task is to open the HTML file in your text editor and in your web browser:

- **Text editor:** Run the Open command (almost always by choosing File ➪ Open or by pressing Ctrl+O or ⌘+O) to open the HTML file you want to work with (if it's not open already, that is).
- **Web browser:** Run the Open command to load the same HTML file that you have open in your text editor. Finding the Open command is either trivial or tricky, depending on your operating system:
  - **Windows:** Whether you're using Edge, Chrome, or Firefox, press Ctrl+O to display the Open dialog box; then select the HTML file and click Open.
  - **macOS:** Whether you're using Safari, Chrome, or Firefox, choose File ➪ Open File (or press ⌘+O) to display the Open dialog; then select the HTML file and click Open.

With your HTML file open in both your text editor and your web browser, here's the basic cycle you use to build your pages:

1. **Add some text and HTML stuff (I define what this "stuff" is in the rest of this chapter) to your file.**
2. **Run the editor's Save command (almost always by choosing File ➪ Save or by pressing Ctrl+S or ⌘+S) to save your work.**
3. **Run the web browser's Reload command. Again, how you invoke Reload depends on the operating system:**
   - **Windows:** Whether you're using Edge, Chrome, or Firefox, press Ctrl+R to reload the page.
   - **macOS:** If you're using Safari or Chrome, choose View ➪ Reload Page (or press ⌘+R) to reload the page. In Firefox, press ⌘+R.

   The web browser reloads the page and displays whatever changes you made to the HTML file in Step 1.
4. **Repeat Steps 1 through 3 as often as you like.**

Getting to Know HTML

Building a web page from scratch using your bare hands may seem like a daunting task. It doesn't help that the codes you use to set up, configure, and format a web page are called the HyperText Markup Language (HTML for short), a name that could only warm the cockles of a geek's heart. Here's a mercifully brief look at each term:

- **HyperText:** An oblique reference to the links that are the defining characteristic of web pages
- **Markup:** The instructions that specify how the content of a web page should be displayed in the web browser
- **Language:** The set of codes that comprise all the markup possibilities for a page

But even though the name HTML is intimidating, the codes used by HTML aren't even close to being hard to learn. There are only a few of them, and in many cases they even make sense!

Working with HTML elements and tags

At its most basic, HTML is nothing more than a collection of markup codes — called *elements* — that specify the structure of your web page. For most of your HTML chores, you create a kind of container. What types of things can reside in this container? Mostly text, although often they will be entire chunks of the web page and even other elements.

Most HTML containers use the following generic format:

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

What you have here are a couple of codes that define the container for a particular HTML element. Many elements are one- or two-letter abbreviations, but sometimes they're entire words. You always surround these elements with angle brackets <>; the brackets tell the web browser that it's dealing with a chunk of HTML and not just some random text. An element surrounded by angle brackets is called a *tag.*

REMEMBER

An HTML code by itself is called an *element*; the element surrounded by angle brackets is known as a *tag.*

The first of these codes — `<element>` — is called the *start tag* and it marks the opening of the container; the second of the codes — `</element>` — is called the *end tag* and it marks the closing of the container. (Note the extra slash (`/`) that appears in the end tag.)

In between the start and end tags, you have the `content`, which refers to whatever is contained in the tag. For example, suppose you want to include in your page the sentence *This is a web page with something important to say,* and you want to punch this up a bit by emphasizing *important.* In HTML, the element for emphasis is `em`, so you'd type your sentence like so:

```
This is a web page with something <em>important
  </em> to say.
```

Notice how I've surrounded the word `important` with `<em>` and `</em>`? All web browsers display emphasized text in italics, so that's how the word appears, as shown in Figure 1-1.

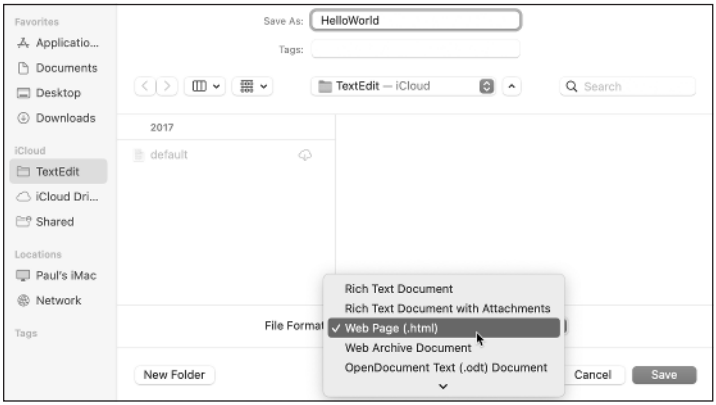

FIGURE 1-1: The sentence revised to italicize the word *important.*

There are tags for lots of other structures, including important text, paragraphs, headings, page titles, links, and lists. HTML is just the sum total of all these tags.

Dealing with tag attributes

You modify how a tag works by adding one or more *attributes* to the start tag. Most attributes use the following generic syntax:

```
<tag attribute="value">
```

Here, you replace *attribute* with the name of the attribute you want to apply to the tag, and you replace *value* with the value you want to assign the attribute, surrounded by either double or single quotation marks. If you add two or more attributes to a tag, be sure to separate each with a space.

For example, you use the `<a>` tag to create a link to another page and within the `<a>` tag you use the `href` attribute to specify the page address. Here's an example:

```
Go to <a href="rutabagas.html">my rutabagas page</a>
```

A barebones HTML page

In this section, I show you the tags that serve as the basic blueprint you'll use for all your web pages.

Your HTML files will always lead off with the following tag:

```
<!DOCTYPE html>
```

This tag (it has no end tag) is the so-called *Doctype declaration*, and it has an eye-glazingly abstruse technical meaning that, happily, you can safely ignore. All I'll say about it is that you have to include this tag at the top of all your HTML files to make sure that your pages render properly. (Also, I tend to write `DOCTYPE` in uppercase letters out of habit, but writing it as `doctype` is perfectly legal.)

Next up, you add the `<html lang="en">` tag. This tag doesn't do a whole lot except tell any web browser that tries to read the file that it's dealing with a file that contains HTML doodads. It also uses the `lang` attribute to specify the document's language, which in this case is English.

Similarly, the last line in your document will always be the corresponding end tag: `</html>`. You can think of this tag as

the HTML equivalent for "The End." So, each of your web pages will include this on the second line:

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

and this on the last line:

```
</html>
```

The next items serve to divide the page into two sections: the head and the body. The head section is like an introduction to the page. Web browsers use the head to glean various types of information about the page. A number of items can appear in the head section, but the only one that makes any real sense at this early stage is the title of the page, which I talk about in the next section, "Giving your page a title."

To define the head, add `<head>` and `</head>` tags immediately below the `<html>` tag you typed in earlier. So, your web page should now look like this:

```
<!DOCTYPE html>
<html lang="en">
    <head>
    </head>
</html>
```

REMEMBER

Although technically it makes no difference if you enter your tag names in uppercase or lowercase letters, the HTML powers-that-be recommend HTML tags in lowercase letters, so that's the style I use in this book, and I encourage you to do the same.

REMEMBER

Notice that I indented the `<head>` and `</head>` tags a bit (by four spaces, actually). This indentation is good practice whenever you have HTML tags that reside within another HTML container because it makes your code easier to read and easier to troubleshoot.

While you're in the head section, here's an added head-scratcher:

```
<meta charset="utf-8">
```

You place this element between the <head> and </head> tags (indented another four spaces for easier reading). It tells the web browser that your web page uses the UTF-8 character set, which you can mostly ignore except to know that UTF-8 contains almost every character (domestic and foreign), punctuation mark, and symbol known to humankind.

The body section is where you enter the text and other fun stuff that the browser will actually display. To define the body, place <body> and </body> tags after the head section (that is, below the </head> tag):

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
    </head>
    <body>
    </body>
</html>
```

WARNING

A common page error is to include two or more copies of these basic tags, particularly the <body> tag. For best results, be sure you use each of these five basic structural tags — <!DOCTYPE>, <html>, <head>, <meta>, and <body> — only one time on each page.

Giving your page a title

When you surf the web, you've probably noticed that your browser displays some text in the current tab. That tab text is the web page title, which is a short (or sometimes long) phrase that gives the page a name. You can give your own web page a name by adding the <title> tag to the page's head section.

To define a title, surround the title text with the <title> and </title> tags. For example, if you want the title of your page to be "My Home Sweet Home Page," enter it as follows:

```
<title>My Home Sweet Home Page</title>
```

Note that you always place the title inside the head section, so your basic HTML document now looks like this:

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>My Home Sweet Home Page</title>
    </head>
    <body>
    </body>
</html>
```

Figure 1-2 shows this HTML file loaded into a web browser. Notice how the title appears in the browser's tab bar.

FIGURE 1-2: The text you insert into the `<title>` tag shows up in the browser tab.

Here are a few things to keep in mind when thinking of a title for your page:

- Be sure your title describes what the page is all about.
- Don't make your title too long. If you do, the browser may chop it off because the tab doesn't have enough room to display it. Fifty or 60 characters are usually the max.
- Use titles that make sense when someone views them out of context. For example, if someone really likes your page, that person may add it to their list of favorites or bookmarks. The browser displays the page title in the Favorites list, so it's important that the title makes sense when a viewer looks at the bookmarks later on.
- Don't use cryptic or vague titles. Titling a page "Link #42" or "My Web Page" may make sense to you, but your visitors will almost certainly be scratching their heads.

Adding some text

Now it's time to put some flesh on your web page's bones by entering the text you want to appear in the body of the page. For the most part, you can type the text between the `<body>` and `</body>` tags, like this (refer to bk01ch01/example04.html):

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>My Home Sweet Home Page</title>
    </head>
    <body>
        Hello HTML World!
    </body>
</html>
```

Figure 1-3 shows how a web browser displays this HTML.

FIGURE 1-3: Text you add to the page body appears in the browser's content window.

Before you start typing, however, there are a few things you should know:

- You may think you can line things up and create some interesting effects by stringing together two or more spaces. Ha! Web browsers chew up all those extra spaces and spit them out into the nether regions of cyberspace. Why? Well, the philosophy of the web is that you can use only HTML elements to lay out a document. So, a run of multiple spaces (or *white space,* as it's called) is ignored.
- Tabs also fall under the rubric of white space. You can enter tabs all day long, but the browser ignores them completely.
- Browsers also like to ignore the carriage return. It may sound reasonable to the likes of you and me that pressing Enter (or

Return on a Mac) starts a new paragraph, but that's not so in the HTML world.

- If you want to separate two chunks of text, you have multiple ways to go, but here are the two easiest:
  - **For no space between the texts:** Place a `<br>` (for line break) tag between the two bits of text.
  - **For some breathing room between the texts:** Surround each chunk of text with the `<p>` and `</p>` (for paragraph) tags.
- If HTML documents are just plain text, does that mean you're out of luck if you need to use characters such as © and €? Luckily, no. For the most part, you can just add these characters to your file. However, HTML also has special codes for these kinds of characters.
- If, for some reason, you're using a word processor instead of a text editor, know that it won't help to format your text using the program's built-in commands. The browser cheerfully ignores even the most elaborate formatting jobs because browsers understand only HTML (and CSS and JavaScript). And besides, a document with formatting is, by definition, not a pure text file, so a browser may have trouble loading it.

Getting Familiar with CSS

One of the things that makes web coding with HTML so addictive is that you can slap up a page using a few basic tags and find that it usually works pretty good when you examine the result in the browser. A work of art it's not, but it won't make your eyes sore. The browsers' default formatting means that even a basic page looks reasonable, but I'm betting you're reading this book because you want to shoot for something more than reasonable. In this section, you discover that the secret to creating great-looking pages is to override the default browser formatting with your own. You do that by augmenting your pages with some CSS.

Understanding cascading style sheets

If you want to control the look of your web pages, the royal road to that goal is a web-coding technology called *cascading style sheets*, or *CSS*. Before getting to the specifics, I answer three simple questions: What's a style? What's a sheet? What's a cascade?

Styles: Bundles of formatting options

In a nutshell, a *style* is a bundle of formatting options rolled into one nice, neat package. That is, it enables you to define a series of formatting options for a given page element, such as a tag like `<div>` or `<h1>`.

Sheets: Collections of styles

The term *style sheet* harkens back to the days of yore when old-timey publishing firms would keep track of their preferences for things like typefaces, type sizes, margins, and so on. All these so-called "house styles" were stored in a manual known as a *style sheet*. On the web, a style sheet is similar: It's a collection styles that you can apply to a web page.

Cascading: How styles propagate

The "cascading" part of the name *cascading style sheets* is a bit technical, but it refers to a mechanism that's built into CSS for propagating styles between elements.

Getting comfy with CSS rules and declarations

Before I show you how to actually use CSS in your web pages, take a second to get a grip on just what a style looks like.

The simplest case is to apply a single formatting option to an element. Here's the general syntax for this:

```
element {
    property: value;
}
```

Here, *element* is a reference to the web page doodad to which you want the style applied. This reference is often a tag name (such as

h1 or div), but CSS has a powerful toolbox of ways you can reference things, which I discuss in Chapter 6.

The *property* part is the name of the CSS property you want to apply. CSS offers a large collection of properties, each of which is a short, alphabetic keyword, such as font-family for the typeface, color for the text color, and border-width for the thickness of a border. The property name is followed by a colon (:), a space for readability, the *value* you want to assign to the property, and then a semicolon (;). This line of code is known in the trade as a *CSS declaration.*

REMEMBER

Always enter the *property* name using lowercase letters. If the *value* includes any characters other than letters or a hyphen, you need to surround the value with quotation marks (single or double).

Notice, too, that the declaration is surrounded by braces ({ and }). You can place multiple declarations between the braces, and that collection is known as a *declaration block*. A declaration block applied to a page item (such as an HTML element) is called a *style rule*.

For example, the following rule applies a 72-pixel (indicated by the px unit) font size to the <h1> tag:

```
h1 {
    font-size: 72px;
}
```

The following example shows a rule with multiple declarations:

```
h1 {
    border-width: 1px;
    border-style: solid;
    border-color: black;
    font-family: Verdana;
    font-size: 72px;
    text-align: center;
}
```

Besides applying multiple styles to a single item, it's also possible to apply a single style to multiple items. You set up the style

in the usual way, but instead of a single item at the beginning of the rule, you list all the items that you want to style, separated by commas:

```
header,
aside,
footer {
    background-color: yellow;
}
```

Applying styles to a page

With HTML tags, you just plop the tag where you want it to appear on the page, but styles aren't quite so straightforward. In fact, there are three main ways to get your web page styled: inline styles, internal style sheets, and external style sheets.

Inserting inline styles

An *inline style* is a style rule that you insert directly into whatever tag you want to format. Here's the general syntax to use:

```
<element style="property1: value1; property2:
  value2; ...">
```

That is, you add the `style` attribute to your tag, and then set it equal to one or more declarations, separated by semicolons.

For example, to apply 72-pixel type to an `<h1>` heading, you could add an inline style that uses the `font-size` CSS property:

```
<h1 style="font-size: 72px;">
```

Embedding an internal style sheet

For easier maintenance of your styles, and to take advantage of the many ways that CSS offers to apply a single style rule to multiple page items, you need to turn to style sheets, which can be either internal (as I discuss here) or external (as I discuss in the next section).

An *internal style sheet* is a style sheet that resides within the same file as the page's HTML code. Specifically, the style sheet is embedded between the `<style>` and `</style>` tags in the page's head section, like so:

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <style>
            Your style rules go here
        </style>
    </head>
    <body>
...
```

Here's the general syntax to use:

```
<style>
    itemA {
        propertyA1: valueA1;
        propertyA2: valueA2;
        ...
    }
    itemB {
        propertyB1: valueB1;
        propertyB2: valueB2;
        ...
    }
    ...
</style>
```

As the preceding code shows, an internal style sheet consists of one or more style rules embedded within a `<style>` tag, which is why an internal style sheet is also sometimes called an *embedded style sheet.*

In the following code, I apply border styles to the `h1` and `h2` elements: solid and dotted, respectively. Figure 1-4 shows the result.

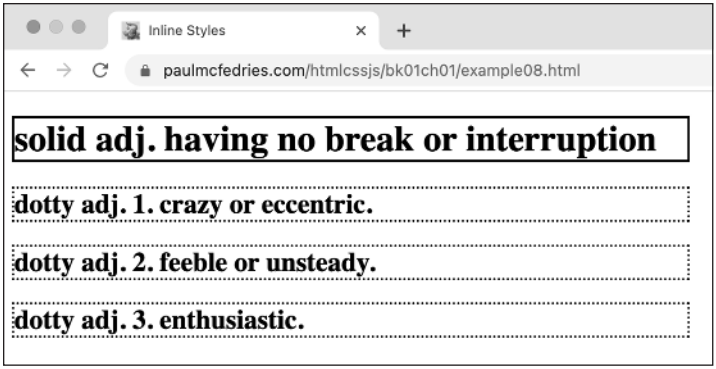

FIGURE 1-4: An internal style sheet that applies different border styles to the `h1` (top) and `h2` elements.

HTML:

```
<h1>solid adj. having no break or interruption</h1>
<h2>dotty adj. 1. crazy or eccentric.</h2>
<h2>dotty adj. 2. feeble or unsteady.</h2>
<h2>dotty adj. 3. enthusiastic.</h2>
```

CSS:

```
<style>
    h1 {
        border-width: 2px;
        border-style: solid;
        border-color: black;
    }
    h2 {
        border-width: 2px;
        border-style: dotted;
        border-color: black;
    }
</style>
```

Note, in particular, that my single style rule for the `h2` element gets applied to all the `<h2>` tags in the web page. That's the power of a style sheet: You need only a single rule to apply one or more style declarations to every instance of a particular element.

The internal style sheet method is best when you want to apply a particular set of style rules to just a single web page. If you have rules that you want applied to multiple pages, you need to go the external style sheet route.

Linking to an external style sheet

Style sheets get insanely powerful when you use an *external style sheet*, which is a separate file that contains your style rules. To use these rules within any web page, you add a special `<link>` tag inside the page head. This tag specifies the name of the external style sheet file, and the browser then uses that file to grab the style rules.

Here are the steps you need to follow to set up an external style sheet:

1. **Use your favorite text editor to create a shiny new text file.**
2. **Add your style rules to this file.**

   Note that you don't need the `<style>` tag or any other HTML tags.
3. **Save the file.**

   It's traditional to save external style sheet files using a `.css` extension (for example, `styles.css`). You can either save the file in the same folder as your HTML file, or you can create a subfolder (named, say, `css` or `styles`).
4. **For every page in which you want to use the styles, add a `<link>` tag inside the page's head section.**

   Here's the general format to use (where *`filename`*`.css` is the name of your external style sheet file):

   ```
   <link rel="stylesheet" href="filename.css">
   ```

   If you created a subfolder for your CSS files, be sure to add the subfolder to the `href` value (for example, `href="styles/`*`filename`*`.css"`).

For example, suppose you create a style sheet file named `styles.css`, and that file includes the following style rules:

```
h1 {
    color: blue;
}
p {
    font-size: 24px;
}
```

You then refer to that file by using the `<link>` tag, as shown here:

```
<!DOCTYPE html>
<html lang="en">
    <head>
       <link rel="stylesheet" href="styles.css">
    </head>
    <body>
        <h1>This Heading Will Appear Blue</h1>
        <p>This text will be displayed in a
  24-pixel font.</p>
    </body>
</html>
```

Why is this use of a style sheet so powerful? You can add the same `<link>` tag to any number of web pages and they'll all use the same style rules, making it a breeze to create a consistent look and feel for your site. And if you decide that your `<h1>` text should be green instead, all you have to do is edit the style sheet file (`styles.css`). Automatically, every single one of your pages that link to this file will be updated with the new style!