CHAPTER **1**

# Introduction to Web Programming

## CHAPTER OBJECTIVES

- Learn the basics of creating a website.
- Learn the basics of HTML—elements, tags, attributes.
- Use structural elements (`html`, `head`, `body`) to form the framework of a web page.
- Fill in a `head` container with `title` and `meta` elements.
- Fill in a `body` container with `h1`, `hr`, `p`, `br`, and `div` elements.
- Learn the basics of Cascading Style Sheets.
- Learn how HTML, the Web, and web browsers originated.
- Use the W3C's Markup Validation Service.

© StationaryTraveller/Getty Images

## CHAPTER OUTLINE

## 1.1 Introduction

Have you ever perused the Web and wondered how its web pages are made? If so, this book is for you. Actually, even if you haven't thought about how web pages are made, this book can still be for you. All you need is a logical mind and an interest in creating things. This book takes you on a journey where you learn to create informative, attractive, and interactive web pages. So climb on board and enjoy the ride!

To make this book accessible to readers with little background in computers, we start slowly and build upon what comes earlier in the book. If you come to something that you already know, feel free to skip it. If you already know how to program, you'll probably want to skip some of the programming basics when we get to JavaScript. But rest assured that unless you already know HTML, CSS, and JavaScript, the vast majority of this book's content should be new to you. After all, we want you to get your money's worth from this book.

Let's start with a brief description of *the Web*, which is short for *World Wide Web*. Most people say "Web" instead of "World Wide Web," and we'll follow that convention. The Web is a collection of documents, called *web pages*, that are shared (for the most part) by computer users throughout the world. Different types of web pages do different things, but at a minimum, they all display content on computer screens. By "content," we mean text, pictures, and user input mechanisms like text boxes and buttons. **FIGURE 1.1** shows a typical web page. Note the web page's text, pictures, text boxes, and buttons. Also note the web page's address shown in the figure's address bar. The web page address is the location where the web page resides on the *Internet*. Speaking of the Internet, what is it? It's a collection of several billion computers connected throughout the world. Each web page is stored on one of those computers.

Figure 1.1 shows the "TED Talks" website. To visit it, open a browser (e.g., Google Chrome, Microsoft Edge, and FireFox) and enter the web page address shown in the figure's address bar.

At the start of this book, we'll focus on displaying text, like the "2400+ talks to stir your curiosity" at the top of Figure 1.1. Next, we'll focus on the appearance of displayed content. Then on to organizational constructs, pictures, sound clips, and video clips. Finally, we will focus on implementing user input *controls*. For example, in Figure 1.1, note the text boxes and the

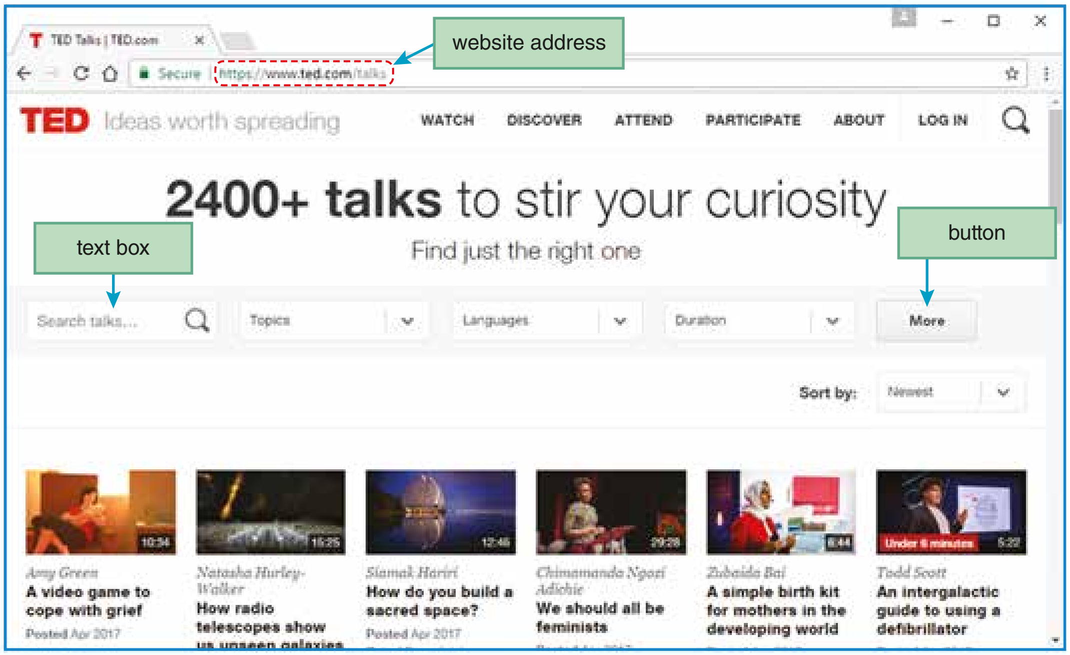

website address
TED Talks | TED.com
Secure https://www.ted.com/talks
TED Ideas worth spreading
WATCH
DISCOVER
ATTEND
PARTICIPATE
ABOUT
LOG IN
2400+ talks to stir your curiosity
Find just the right one
text box
button
Search talks...
Topics
Languages
Duration
More
Sort by: Newest
Amy Green A video game to cope with grief Posted Apr 2017
Natasha Hurley-Walker How radio telescopes show us unseen galaxies
Siamak Hariri How do you build a sacred space? Posted Apr 2017
Chimamanda Ngozi Adichie We should all be feminists
Zubaida Bai A simple birth kit for mothers in the developing world
Todd Scott An intergalactic guide to using a defibrillator
Under 6 minutes
© Ted Conferences, LLC

**FIGURE 1.1** **A typical web page**

buttons. Those are controls. You'll learn about those controls, plus more controls, in the last several chapters.

In this first chapter, we stick with the basics, so you can get up and running quickly. Specifically, we start with some overarching concepts that explain the process of web page development and dissemination. Then, we introduce the basic constructs that you'll use to describe and display a web page's content. Next, we provide a cursory overview of Cascading Style Sheets (CSS), which you'll use to display a web page's content in a pleasing, formatted manner. Finally, we present a brief history of the primary language used to write all web pages—HTML.

## 1.2 Creating a Website

A *website* is a collection of related web pages that are normally stored on a single web server computer. A *web server* is a computer system that enables users to access web pages stored on the web server's computer. The term "web server" can refer to the web page-accessing software that runs on the computer, or it can refer to the computer itself.

To create a website, you'll need these things: (1) a text editor, (2) an upload/publishing tool, (3) a web hosting service, and (4) a browser. We'll describe them in the upcoming paragraphs.

### Text Editors

There are many different text editors, with varying degrees of functionality. Microsoft's Notepad is free and provides no special web functionality. To use it, the web developer simply enters text, and the text appears as is. Although it's possible to use a plain text editor such as Notepad, most web developers use a fancier type of text editor—a *web authoring tool*. Different web authoring tools have different features that are intended to make the web development process easier. At a minimum, web authoring tools are able to suggest valid code after the user has typed part of a command. This is done by showing a pop-up to the user that suggests valid code that could complete the command currently being entered. This auto-complete mechanism is often called *intellisense* and sometimes called *picklist*. Another feature common to all web authoring tools is *WYSIWYG*, pronounced "wizeewig." It stands for "what you see is what you get." WYSIWYG means that as you're editing your text, you can see what your text will look like after it's eventually uploaded to a website.

On this book's website, we provide a tutorial for learning how to use the Visual Studio web authoring tool. Visual Studio is from Microsoft, so it's not free. But fear not, Microsoftophiles. If you plan to use Visual Studio for nonbusiness purposes, you can download Microsoft Visual Studio Community for free (that means you—students, faculty, and open-source project contributors). Visual Studio Community includes all the functionality of Visual Studio's professional version.[1]

There are a lot of other web authoring tools that you are welcome to learn on your own. Visual Studio and its offshoots run on Windows, but if you have a Mac(intosh) computer, check out Adobe's Dreamweaver web authoring tool. It works on both Windows and Mac. Or, do a Google search for other web authoring tools—most are free and some are quite good!

Normally, web authoring tools enable developers to create not just web pages, but other software as well. Such general-purpose web authoring tools are normally referred to as *integrated development environments*, or *IDE*s for short.

## Web Page Uploads

After you enter your web page text on your local computer with your favorite IDE, you'll probably want to *publish* it. Publishing means that you upload your web page to a web server computer so other users can access it on the Web. Some IDEs, like Dreamweaver, provide built-in uploading capabilities, but other IDEs, like Visual Studio, do not. For IDEs that do not provide built-in uploading capabilities, you'll need to use a separate file upload tool. There are lots of file upload tools. On this book's website, we provide a tutorial for learning how to install and use a free and intuitive file upload tool called WinSCP.

## Web Hosting Service

For a file upload tool such as WinSCP to work, you need to have a web server computer on which to store the uploaded files. For the uploaded files to be accessible as web pages on the Web, your web server computer needs to have a web hosting service in place. The web developer usually doesn't have to worry about the web hosting service software. If the web developer is part of a medium- to large-sized organization, then the organization's information technology (IT)

[1] "Visual Studio Community," *Microsoft*, https://www.visualstudio.com/vs/community/.

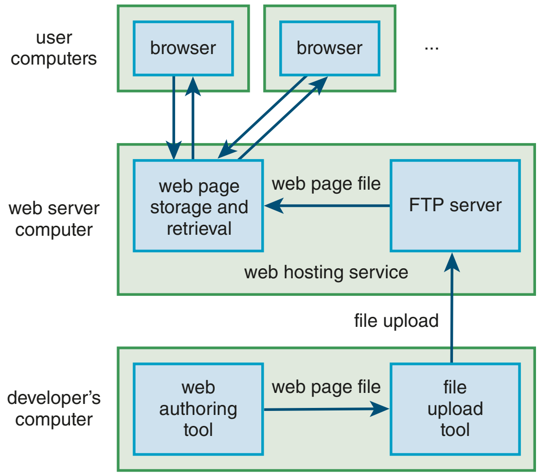


**FIGURE 1.2 Website file processing**

department will install and maintain the web hosting service. On the other hand, if the web developer is part of a very small organization or not part of an organization at all, the developer will need to set up the web hosting service or rely on a generic web-hosting company (e.g., GoDaddy.com) to do so. Regardless of who's in charge of the web hosting service, all web hosting services need to have a mechanism for receiving uploaded files from a file upload tool. Typically, that mechanism is an FTP (file transfer protocol) server, which is a program that runs on the web server computer. **FIGURE 1.2** is a pictorial description of how the FTP server fits in with the rest of the website creation process.

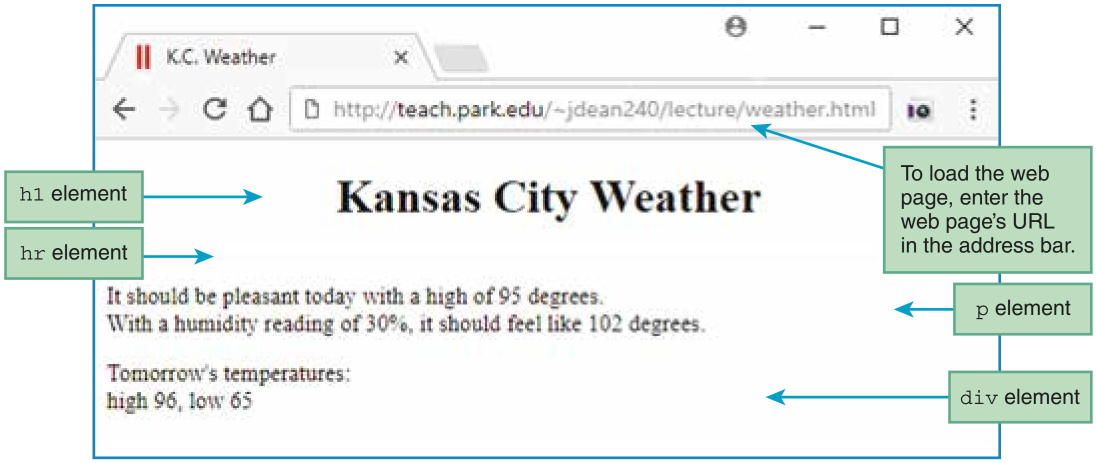
### Browsers

The top of Figure 1.2 shows the final part of the website experience: browser access. A *browser* is a piece of software that enables a user to retrieve and view a web page. According to http://gs.statcounter.com, the most popular browsers for computers are Google Chrome, Microsoft's browsers (Microsoft Edge and Internet Explorer), and Mozilla[2] Firefox, with Google Chrome at #1. Other browsers are Safari (for Mac devices), Opera, and Android's default browser. Safari and Android are particularly popular with mobile devices.

## 1.3 Web Page Example

Note **FIGURE 1.3**, which shows a simple Kansas City Weather web page. Before showing you the behind-the-scenes code for the Kansas City Weather page, we first need to go over some preliminary concepts. We'll start with the website address. Formally, the website address value is known

[2] The name "Mozilla" comes from a combination of the words Mosaic (the first popular graphical browser) and Godzilla (the first known sea monster formed by nuclear radiation).


K.C. Weather
http://teach.park.edu/~jdean240/lecture/weather.html
h1 element
hr element
p element
div element
To load the web page, enter the web page's URL in the address bar.

Kansas City Weather

It should be pleasant today with a high of 95 degrees.
With a humidity reading of 30%, it should feel like 102 degrees.

Tomorrow's temperatures:
high 96, low 65

**FIGURE 1.3 Kansas City Weather web page**

as a *URL*, which stands for *Uniform Resource Locator*. That name is not all that intuitive, so just remember that a URL is a website address. Here's the URL for the Kansas City Weather page in Figure 1.3:

```
http://teach.park.edu/~jdean240/lecture/weather.html
```

The `http` refers to the *hypertext transfer protocol*, where a protocol is a set of rules and formats for exchanging messages between computers. After `http` comes a delimiter, `://`, and then the name of the web server computer that stores the web page. For this example, the web server computer is `teach`. Next comes the domain that describes how the web server can be found on the Internet. For this example, the domain is `park.edu`. Next, there's a sequence of directories and subdirectories (also called folders and subfolders) that indicate where the web page is stored on the web server computer. That's called the *path*. For this example, the path is `~jdean240/lecture`. The ~ (tilde) at the left indicates that the directory is a home directory for a user's account. There's no requirement that you use a ~ for a user's home directory. It's a standard convention at universities, where users (students and teachers) like to do their own thing, but most businesses do not use ~'s. In the example, after the `~jdean240` home directory, there's a / and then `lecture`. The term `lecture` is a subdirectory of the home directory, and the / is a delimiter that separates the home directory from the subdirectory.

In the example, after the `lecture` subdirectory, there's a / and then `weather.html`. The phrase `weather.html` is the web page's filename, and the / is another delimiter. This time, the / separates the subdirectory from the web page filename.

In Figure 1.3, all the things you see below the address bar are web page *elements*—`h1`, `hr`, `p`, and `div`. We'll have more to say about those elements later, but here's just a brief introduction for now.

The `h1` element is used to implement a web page heading, with the "h" in `h1` standing for "heading." The `hr` element is used to implement a horizontal line, with the "h" and "r" standing for "horizontal" and "rule," respectively. The `p` element is used to implement a paragraph. Finally, a `div` element is used to group words together as part of a division within a web page.

There are additional elements used to implement the Kansas City Weather page, but they're not as intuitive as the elements we've described. For example, there's a `body` element that forms the entire white area under the address bar. Coming up, we'll show how to implement that element, plus all the other elements in the Kansas City Weather page. But first a tribute to the central role of elements to a web page.

A web page's elements hold the web page's content, which is the most important part of a web page. After deciding on which elements to use and implementing those elements, you'll want to focus on formatting the content. In Chapter 3, we'll describe how to format the content using *CSS*. Optionally, you can add behaviors for some or all the elements. Later in the book, we'll describe how to add behaviors using JavaScript. For example, in the Kansas City Weather page, you could add a behavior to the `h1` element, so that when you click it, the subsequent paragraph turns blue. A rather odd scenario, but you get the idea.

## 1.4 HTML Tags

Now we're going to describe how to implement elements for a web page. To implement an element for a web page, you'll need to use tags. For example, if you want to implement a `strong` element (in order to put emphasis on the element's content and display using boldface), surround the content with `<strong>` tags. Here's how to implement a `strong` element for the word "very":

```
Caiden was <strong>very</strong> happy when her tooth finally came out.
```

The use of tags is the key characteristic of a *markup language*. Why is it called "markup"? A markup language "marks up" a document by surrounding parts of its content with tags. Web pages are implemented with HTML, which stands for *Hypertext Markup Language*. You already know what "markup" means. The "hyper" in "Hypertext" refers to HTML's ability to implement hyperlinks. A *hyperlink* (or *link* for short) is where you click on text or an image in order to jump to another web page. So HTML is a language that supports markup tags for formatting and the ability to jump to other web pages.

To simplify the web page creation process, the WYSIWYG option (for web authoring tools) lets you hide the HTML tags. Unfortunately, in hiding the HTML tags, the developer loses some control, and the resulting web pages tend to be sloppy. Even if your web authoring tool generates clean HTML code when in WYSIWIG mode, we strongly recommend that you enter all your tags explicitly, at least for now. That will help you learn HTML details so you'll be able to understand and debug subtle code issues.

Most, but not all, HTML tags come in pairs with a start tag and an end tag. For example, the following code uses an `<h1>` start tag and an `</h1>` end tag:

```
<h1>Today's Weather</h1>
```

Note that each tag is surrounded by angled brackets, and the end tag starts with "</". The h1 heading tags cause the enclosed text ("Today's Weather") to display like a heading. That usually means that the enclosed text will be boldfaced, large, and surrounded by blank lines. Note the use of the term "usually." The official HTML standard/specification often doesn't specify precise display characteristics. Consequently, not all browsers display tags in the exact same way.

Besides h1, there are other heading elements—h2 through h6. The element h1 generates the largest heading, and h6 generates the smallest. Use a heading tag whenever you want to display a heading above other text. Headings are usually at the top of the page, but they can be in the middle of the page as well.

The first entity inside a tag is the tag's type. In the previous example, the tag's type is h1. In this simple example, there's only one entity inside the tags—the tag's type. Later, we'll see examples where there are additional entities inside the tags.

When a tag is discussed in general, without reference to a particular tag instance, it is called an element and it is written without the angled brackets. For example, when discussing the <h1> tag in general, refer to it as the h1 element.

There are two types of elements—container elements and void elements. A *container element* (usually called simply a "container") has a start tag and an end tag, and it contains content between its two tags. For example, the h1 element is a container. On the other hand, a *void element* has just one tag, and its content is stored within the tag. We'll see an example of that when we get to the meta element.

## 1.5 Structural Elements

Take a look at **FIGURE 1.4**. It shows the HTML source code that was used to generate the Kansas City Weather web page shown earlier. What's source code, you ask? With many programming languages, two types of code are associated with a single program. There's *source code*, which the programmer enters, and there's *executable code*, which is low-level code that the computer hardware understands and executes (runs). Executable code is generated from the source code with the help of something called a *compiler*. As you learn HTML, there's no need to worry about executable code because it is generated automatically behind the scenes and you never see it. So, why did we bring it up? So you can mention executable code in conversation and impress your techie friends? Yes, there's always that, but also, HTML has source code, and to understand the term source code, it's helpful to understand the term executable code. HTML code is source code because HTML code comes directly from the programming source—the programmer.

Next, we'll describe the different sections of code in Figure 1.4. Let's start with the really important HTML constructs that you'll use as the basic framework for all your web pages—doctype, html, head, and body. The doctype construct is considered to be an instruction, not an element, and it goes at the top of every web page. The html, head, and body elements form the basic structure of a web page, so we'll refer to those elements as *structural elements*. Be aware that that's not a standard term. We made it up because it will make future explanations easier. The doctype instruction plus the structural elements form the skeleton code shown in **FIGURE 1.5**. You can use that skeleton code for all your web pages.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<meta name="author" content="John Dean">
<meta name="description" content="Kansas City weather conditions">
<title>K.C. Weather</title>
<style>
  h1 {text-align: center;}
  hr {width: 75%;}
</style>
</head>

<body>
<h1>Kansas City Weather</h1>
<hr>
<p>
  It should be pleasant today with a high of 95 degrees.<br>
  With a humidity reading of 30%, it should feel like 102 degrees.
</p>
<div>
  Tomorrow's temperatures:<br>
  high 96, low 65
</div>
</body>
</html>
```

**FIGURE 1.4 Source code for Kansas City Weather web page**

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

<body>
  ⋮
</body>
</html>
```

**FIGURE 1.5 Skeleton code using just doctype and the structural elements**

The first construct, `<!DOCTYPE html>`, tells the browser what type of document the web page is. Its `html` value (in `<!DOCTYPE html>`) indicates that the document is an HTML document, and more specifically that the document uses the HTML5 standard for its syntax. *Syntax* refers to the words, grammar, and punctuation that make up a language.

After the doctype instruction comes the html element. It's a container, and it contains/surrounds the rest of the web page. Its start tag includes lang="en", which tells the browser that the web page is written in English. The head and body elements are also containers. The head element surrounds elements that provide information associated with the web page as a whole. The body element surrounds elements that display content in the web page. Container elements must be properly *nested*, meaning that if you start a container inside another container, you must end the inner container before you end the outer container. Because the body element starts inside the html element, the </body> end tag must come before the </html> end tag. In Figure 1.5, note how the head and body elements are properly nested within the html element.

## 1.6 title Element

Let's now dig a little deeper into the head element. The head code in **FIGURE 1.6** comes from the weather page, but in the interest of keeping things simple, we've omitted its style element. The style element is more complicated, and we'll discuss it later in this chapter.

The head element contains two types of elements—meta and title. In your web pages, you should position them in the order shown in Figure 1.6, meta and then title. But in the interest of clarity, we'll discuss the title element first.

Remember that the head element surrounds elements associated with the web page as a whole. The web page's title pertains to the entire web page, so its title element goes within the head container. The title element's contained data (e.g., "K.C. Weather") specifies the label that appears in the browser window's *title bar*. Go back to Figure 1.3 and verify that "K.C. Weather" appears in the tab near the top of the window. With browsers that support tabbed windows, that tab is considered to be the browser's title bar.

The official HTML standard requires that every head container contains a title element. Besides providing a label for your browser window's title bar, what's the purpose of the title element? (1) It provides documentation for someone trying to maintain your web page, and (2) it helps web search engines find your web page. In case you haven't heard of a *web search engine*, it's software that searches the Internet for user-specified information and returns a list of links to web pages that are likely to contain the requested information. Google is the preeminent search

```
<head>
<meta charset="utf-8">
<meta name="author" content="John Dean">
<meta name="description" content="Kansas city weather conitions">
<title>K.C. Weather</title>
</head>
```

**FIGURE 1.6 head container for Kansas City Weather web page**

engine, as evidenced by the use of "google" as a verb. For example, "Did the CIA make up dinosaurs?" "Not sure. I'll google it."

## 1.7 `meta` Element

In Figure 1.6, note the `meta` elements within the `head` container. The `meta` elements provide information about the web page. If you look up "meta" in a standard dictionary, you'll probably see a confusing definition. In everyday speech and in HTML, "meta" means "about itself." As in "Check out this video of two guys watching a how-to video about appreciating videos." "Dude, stop now. You're hurting my brain. That's so meta!"

There are many different types of `meta` elements—some you should always include, but most are just optional. We'll present a few of the more important ones soon, but first some details about the `meta` element's syntax.

The `meta` element is a void element (not a container), so it does not have an end tag. Note how there are no end tags for the three `meta` elements in Figure 1.6. Many web programmers end their void elements with a space and a slash. For example:

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

The space and slash are a bit outdated, and we recommend omitting them. If you decide to include them, be consistent. Don't worry about the meaning of `charset` and `utf-8` for now; we'll discuss them shortly.

## 1.8 HTML Attributes

Before we formally introduce a few of the different types of `meta` elements, we first need to discuss attributes, which are used in all `meta` elements. Container elements provide information between their start and end tags. Void elements (including the `meta` element) have no end tags, so they can't provide information that way. Instead, they provide information using attributes. In the following example, `charset` is an attribute for a `meta` element:

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

Most attributes have a value assigned to them. In this example, `charset` is assigned the value `"utf-8"`. Although most attributes have a value assigned to them, some do not. Later on, we'll see some attributes that appear by themselves, without a value. You should always surround attribute values with quotes, thus forming a string. A *string* is a group of zero or more characters surrounded by a pair of double quotes (") or a pair of single quotes ('), with double quotes preferred.

Attributes are more common with void elements, but they can be used with container elements as well. Here's an example of a container element that uses an attribute:

```
<html lang="fr">
  •
  •
  •
</html>
```

The lang attribute tells the browser that the element is using a particular language for its content. You can use the lang attribute for any element. Here we're using it for the html element, so it means that the entire web page uses French. You're not required to use the lang attribute, but we recommend that you do include it for the html element. For web pages written in English, use `<html lang="en">`.

Why would you want to specify an element's language? The W3C's Internationalization Activity group (https://www.w3.org/International/questions/qa-lang-why.en) provides quite a few good reasons, and here are a few of them:

- Help search engines find web pages that use a particular language.
- Help spell-checker and grammar-checker tools work more effectively.
- Help browsers use appropriate fonts.
- Help speech synthesizers pronounce words correctly (we'll discuss speech synthesizers in Chapter 5).

If these benefits haven't convinced you to use the lang attribute, consider the fact that as technology has improved, the lang attribute's usefulness has grown over the years. That trend will undoubtedly continue. It's hard to know what cool things the lang attribute might help with in the future, and you should plan for those cool things by including the lang attribute now.

## meta charset Element

Now that you've learned syntax details for element attributes, it's time to focus on semantic details. Syntax refers to the punctuation rules for code. *Semantics* refers to the meaning of the code. First up—the semantics for the meta charset element.

When a web server transmits a web page's source code to an end-user's computer, the web server doesn't transmit the source code's characters the way you see them in this book or on a keyboard. Instead, it transmits coded representations of the source code's characters. The coded representations are in *binary*, which means a sequence of 0's and 1's, where each 0 and 1 is a *bit* (so 10110011 is a binary sequence of 8 bits). There are different encoding schemes, and in order for the receiving end of a transmission to understand the transmitted binary data, the receiver has to know the encoding scheme used by the sender. For web page transmissions, the meta charset element specifies the encoding scheme. Normally, you should use a charset value of "utf-8" because all modern browsers understand that value.[3] The encoding scheme is sometimes referred

[3] UTF-8 (UCS Transformation Format—8-bit) is a variable-width encoding that can represent every character in the Unicode character set. It encodes each Unicode value using one to four 8-bit bytes.

to as a character set, and that's what charset stands for. If you omit the meta charset element, your web page will usually work because most browsers assume UTF-8 encoding by default. But don't count on the default. In omitting the meta charset element, you run the risk of characters being interpreted incorrectly.

### meta name Element

Most of the meta elements use the name attribute to specify the type of information that's being provided. Common values for the meta name attribute are author, description, and keywords. Here's an example with an author value for a name attribute:

```
<meta name="author" content="John Dean">
```

The name and content attributes go together. The name attribute's value specifies the type of thing that the content attribute's value specifies. So in this example, with the name attribute specifying "author," the content attribute specifies the author's name ("John Dean"). Why is knowing the author's name important? Often, the person who fixes or enhances a web page is different from the person who originally wrote the web page. By specifying the author, the fixer/enhancer knows whom to ask for help.

In the following examples, the name attribute uses the values "description" and "keywords":

```
<meta name="description" content="Kansas City weather conditions"
<meta name="keywords" content="KC, weather, meteorology, forecast"
```

The meta description element and also the meta keywords element help web search engines find your web page. In addition, the meta description element helps the person reading the code learn the purpose of the web page.

The meta description element isn't as important as the meta author and meta charset elements. Typically, HTML code is straightforward, so unless you've got tricky code, it's OK to omit the meta description tag. Feel free to include it if you feel that it's beneficial. Likewise, the meta keywords element can go either way—include it or omit it.

## 1.9 body Elements: hr, p, br, div

In the prior sections, we covered the elements that appear inside the weather page's head container. Now let's cover the elements that appear inside the weather page's body container. In **FIGURE 1.7**, which shows the body container code, note the h1, hr, p, br, and div elements. We've already talked about the h1 heading element, so now let's focus on the other elements.

The hr element is used to *render* a horizontal line. When a browser renders an element, it figures out how the element's code should be displayed. To keep things simple, you can think of "render" as a synonym for "display." Go back to Figure 1.3 and note the horizontal line on the weather page browser window. The "h" in hr stands for horizontal. The "r" in hr stands for rule, presumably because a rule is another name for a ruler, which can be used to make a straight line. The hr element is a void element, so it uses just one tag, <hr>.

```
<body>
<h1>Kansas City Weather</h1>
<hr>
<p>
  It should be pleasant today with a high of 95 degrees.<br>
  With a humidity reading of 30%, it should feel like 102 degrees.
</p>
<div>
  Tomerrow's temperatures:<br>
  high 96, low 65
</div>
</body>
```

**FIGURE 1.7 `body` container for Kansas City Weather web page**

The `p` element is a container for a group of words that form a paragraph. Normally, browsers will render a `p` element's enclosed text with a blank line above the text and a blank line below it. Go back to Figure 1.3 and note the blank lines above and below the two-sentence paragraph.

In Figure 1.7, note how we indented the text between the `<p>` start tag and the `</p>` end tag. Whenever you've got a `p` element whose enclosed text is greater than one line, you should put the start and end tags on separate lines and indent the enclosed text. That rule is an example of a coding-style convention rule (or style rule for short). Whenever you write a program, including an HTML program, it's important to follow standard coding-style conventions, so your program is easy to read by you and also by future programmers who need to understand your program. Programmers get used to certain conventions, such as when to use uppercase versus lowercase, when to insert blank lines, and when to indent. You don't want to jar someone reading your program by using nonstandard conventions. For a complete list of coding-style conventions used in this book, see the two appendices: Appendix 1, HTML5 and CSS Coding-Style Conventions, and Appendix 2, JavaScript Coding-Style Conventions. You might want to skim the appendices now, but don't worry about the details. We'll cover those details as we proceed through the book.

Even though it's a container, the HTML standard allows `p`'s start tag to appear without its end tag. However, that's considered to be bad style, so don't do it. You should never omit end tags for container elements because then it's more difficult for the browser and for people reading your source code to figure out where the container element ends.

A `div` element is also a container for a group of words, but it's more generic, so the words don't have to form sentences in a paragraph. `div` stands for division because a division can refer to a part of something that has been divided, and a `div` element is indeed a part of a web page. Normally, the `div` element causes its enclosed text to have single line breaks above and below it. If a `div` element's enclosed text is greater than one line, then proper style suggests putting the `<div>` tags on separate lines and indenting the enclosed text.

Except for single line breaks instead of blank lines, the characteristics for a `div` element are the same as for a `p` element. So when should you use a `p` element versus a `div` element? Use a `p` element if the enclosed text forms something that would normally be considered a paragraph. On the other hand, use a `div` element if the enclosed text is related in some way, but the text would not normally be considered a paragraph. If you use the p element only for bona fide paragraphs, then the rest of the web page can process p elements as paragraphs, and you avoid including non-paragraphs in that processing. For example, you could use Cascading Style Sheets (described in the next section) to indent each paragraph's first line.

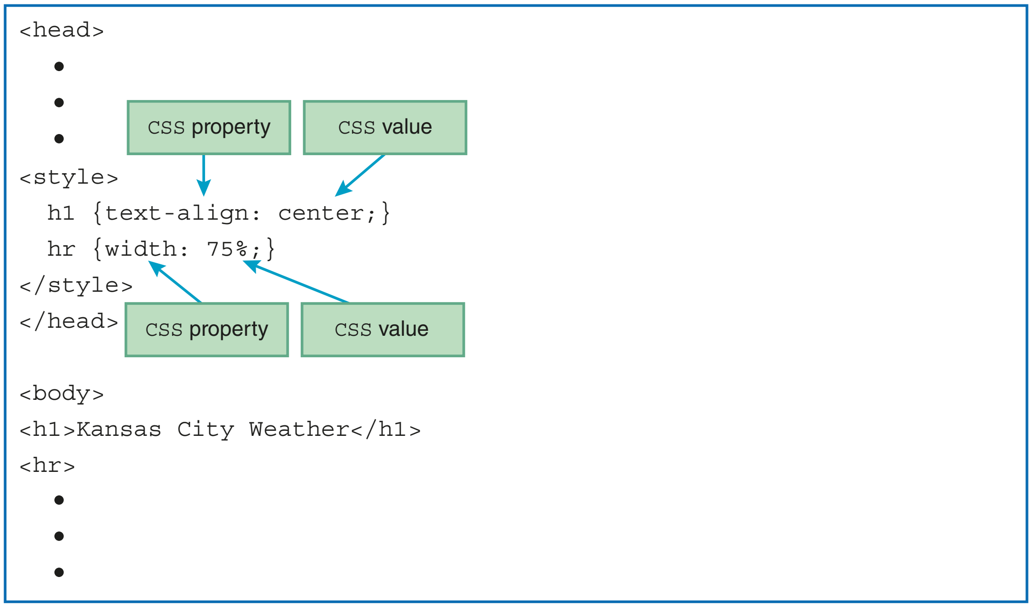
Finally, there's the br element, which is used to render a new line. In Figure 1.7, note this line:

```
It should be pleasant today with a high of 95 degrees.<br>
```

You can see the br element's new line in Figure 1.3's browser window. Specifically, the line "With a humidity reading of 30%, …" starts on a new line even though there is room for it to start at the end of the previous line.

## 1.10 Cascading Style Sheets Preview

We'll wait until Chapter 3 for a robust presentation of Cascading Style Sheets (CSS), but for now it's appropriate to give you a preview so you're aware of the basic concepts. As you may recall, CSS allows you to add formatting to your web pages. The formatting rules go inside a style container. In the skeleton code for the weather web page in **FIGURE 1.8**, note the style container within the head container. Within the style container, note the two lines that begin with h1 and hr. Those lines are the rules that apply to the h1 and hr elements in the body container. Each rule has a CSS property and a CSS value, separated by a colon. The first rule, for h1, uses a text-align property with a value of center. Other text-align values are left and right. The second rule, for hr, uses a width property with a value of 75%.



```
<head>
   •
   •
   •
<style>
  h1 {text-align: center;}
  hr {width: 75%;}
</style>
</head>

<body>
<h1>Kansas City Weather</h1>
<hr>
   •
   •
   •
```

**FIGURE 1.8** **style container for Kansas City Weather web page**

Go back to the web page display in Figure 1.3 and note how `h1` and `hr` are centered. Note how the `hr` element spans 75% of the width of the page. By default, `h1` would be left aligned and the `hr` element would span 100% of the page. Thus, the CSS rules changed the default. If you reduce the `hr` element's width, as in this weather page example, then alignment becomes apparent, and the `hr` element gets center alignment by default. To change its alignment, use `margin-right: 0%` for right alignment or `margin-left: 0%` for left alignment.

## 1.11 History of HTML

In 1989, to help with collaborative research at CERN (the European Laboratory for Particle Physics in Geneva, Switzerland), Tim Berners-Lee came up with the idea of adding "hypertext links" to research papers, so when one paper referred to another, the reader could click the link and quickly go to the other paper. From 1989–1991, Berners-Lee was quite prolific: (1) He designed HTML, with hypertext links as the key feature, (2) he designed the concepts behind the World Wide Web, including the HTTP protocol, and (3) he created a prototype browser for surfing the Internet with HTML web pages. In 1993, Tim Berners-Lee and Dan Connolly submitted the first formal proposal for HTML to the Internet Engineering Task Force (IETF). In 1994, Berners-Lee founded the World Wide Web Consortium (W3C) at the Massachusetts Institute of Technology, with the W3C taking over the stewardship of the HTML standard.

By 1997, the HTML standard evolved to HTML4, with HTML4's last revision appearing in 2000 as HTML 4.01. Bothered by all the poorly formed web pages, the W3C decided to try to force web programmers to conform to stricter syntax rules by introducing XHTML 1.0 Strict in 2000. The "stricter syntax rules" were borrowed from the extensible markup language (XML), and that's what the X in XHTML stands for—XML. XML is not so much a language as a set of rules for how you can define your own language. XHTML 1.0 Strict enforced XML rules such as (1) requiring quoted values for all attributes, (2) requiring a / for all void elements, and (3) requiring an end tag for every container element.

However, even for pages labeled with the new XHTML 1.0 Strict standard, browsers continued to accept poorly formed web pages and render them just fine. Consequently, there was no urgency for programmers to comply with the new XHTML 1.0 Strict standards, and many programmers didn't bother to try to comply.

In 2001, the W3C remedied this browser leniency problem by developing and approving a harsher standard, XHTML 1.1, which specified a new "fail on first error" system. The standard said that if a web page's code does not comply with the standard in any way, the browser should display an error message and not attempt to display the web page's normal content. Internet Explorer (IE), the #1 browser at the time, was unable to render XHTML 1.1 pages. When IE saw the XHTML 1.1 label, it would prompt the user to "Save to disk." Yikes! What a disarmingly worthless message!

Because of this harsh penalty, web programmers avoided using XHTML 1.1, for the most part. The W3C considered XHTML 1.1 to be a stopgap measure that would pave the way to a future standard, XHTML 2.0. XHTML 2.0's goal was to eliminate all problems with past versions of HTML and XHTML. Because XHTML 2.0 development was so slow, and because it was such a departure from what web programmers and browser vendors were used to, some members of the W3C were angered, and in 2004, they left the W3C to make their own new standard. They formed a new working group, *WHATWG*, which stands for *Web Hypertext Application Technology Working Group*.

The primary defectors from the W3C were people from the Mozilla and Opera browser companies. They determined that if XHTML 2.0 came to fruition, they would have to rewrite their browser software. This was doable, but they would end up with a browser that would be incompatible with their customers' existing web pages. Then their customers would defect to other browsers, and with no customers, no money.

The WHATWG spent five years, from 2004 through 2009, creating a Web Apps 1.0 standard. The new standard was a combination of features found in HTML 4.01 and XHTML 1.0, features supported by existing browsers, and new features. Web Apps 1.0 was better than W3C's standards in terms of clearly defining how browsers were supposed to handle errors. Web Apps 1.0 made no attempt to be XML-compliant. It was defined to be more lenient than XHTML; consequently, it was backwards compatible with existing web pages.

In 2009, after making very little progress on XHTML 2.0, the W3C abandoned their XHTML 2.0 efforts and formed a collaborative relationship with the WHATWG, using the Web Apps 1.0 standard as the basis for their current HTML5 standard. In 2011, the W3C came out with a super-cool HTML5 logo, shown here. Display it proudly on all your programmer fashion-wear!

The term "HTML5 standard" is a loose term in that it can refer to any of the W3C's different HTML5 versions. The first version was simply called "HTML5." The next version was called "HTML 5.1," with a space before 5.1. As of 2017, HTML 5.1 was the W3C's official "recommendation," and HTML 5.2 was a "working draft," with the expectation that it would soon become a recommendation. This book presents syntax and semantics from the HTML 5.1 standard, which is a superset of the HTML5 standard. To simplify things, we will use the generic term HTML5 throughout the remainder of the book. If we introduce something that's outside the scope of the original HTML5 standard, we'll point that out, but for the most part, the HTML5 versions are the same.

An indirect benefit of HTML5's rise to prominence is that it has helped to tamp down the *browser wars*. Since the dawn of browsers, to increase their market share, browser manufacturers have added features to their browsers that go above and beyond the HTML standard. This is sometimes a good thing in that end users are treated to cool new features. But from a web programmer's perspective, this is generally a bad thing because programmers are expected to write code that takes advantage of those new features, and writing that code is hard. Specifically, with different users using different browsers and different versions of those browsers, writing and maintaining such *cross-browser-compatible code* is a very time-consuming and messy process. Thus, web programmers refer to the situation disparagingly as the browser wars. In implementing the HTML5 standard, the WHATWG and the W3C incorporated the best added features of the different browsers in order to convince the different browser manufacturers to get on board with the new standard. For the most part, since HTML5's inception, the browser manufacturers have complied with it and have not added as many nonstandard features as in the past. But cross-browser incompatibilities still exist and browser skirmishes continue.

## 1.12 HTML Governing Bodies

As mentioned, the W3C "formed a collaborative relationship with the WHATWG." That was way back in 2009; so did the W3C eventually merge with the WHATWG and form just one governing body? No—there are still two separate organizations even though their roles overlap. They both maintain their own HTML standard. That's a lot of overlap!

The W3C's mission statement says, "To lead the World Wide Web to its full potential by developing protocols and guidelines that ensure the long-term growth of the Web."[4] You can tell from its mission statement that the W3C does a lot more than just maintain its HTML standard. Check out its home page at https://www.w3.org and you'll see a myriad of different tools and articles, all pertaining to the Web. We'll use some of those tools, but for now, we're primarily interested in the HTML standard. More specifically, we're interested in the work done by the W3C's HTML working group (HTMLWG). They publish new versions of the HTML5 standard when they feel their updates are in a stable position.

The WHATWG's mission is narrower in scope than the W3C's mission. WHATWG's home page simply says that they "maintain and evolve HTML." But don't think that they're slackers; maintaining and evolving HTML is a big undertaking. They consider "HTML" to be an umbrella that includes HTML5 (which they refer to as just "HTML"), CSS, and the Document Object Model (DOM), and they provide specifications for each of them. We'll describe each of those technologies as we progress through the book. The W3C also provides specifications for HTML5, CSS, and the DOM. So, what's the difference between the two organizations? The WHATWG's standard is deemed "living," which means the WHATWG is free to make updates at any time, and they don't bother to assign new version numbers to their standard when they do so. That's different from the W3C, which publishes new versions, with version numbers, only after they feel their updates are stable.

Having two HTML standards might seem like a mess, but remember that the Web itself was built organically, with lots of disparate contributors from around the world. The W3C and WHATWG have a vested interest in making sure their standards are pretty close. After all, if one organization goes too far into left field, they'll lose supporters. Some browser manufacturers prefer to follow the W3C's standard because the W3C is a more venerable institution, and they provide stable versions. On the other hand, some browser manufacturers prefer to follow the WHATWG's standard because the WHATWG tends to be more receptive to new trends and suggestions for changes. We'll refer primarily to the W3C standard. But because we use syntax that is common to both standards and supported by all the major browsers, our preference for the W3C standard is pretty much irrelevant. The differences in the standards are prominent only when it comes to the leading-edge stuff.

[4] "W3C Mission," *W3C*, https://www.w3.org/Consortium/mission.html.

It's true that the two organizations work collaboratively, but their relationship isn't necessarily a well-oiled machine. Their relationship is more like siblings who grouse every now and then. For example, note this snippet from a WHATWG web page:[5]

> The W3C publishes some forked versions of these specifications. We have requested that they stop publishing these but they have refused. They copy most of our fixes into their forks, but their forks are usually weeks to months behind. They also make intentional changes, and sometimes even unintentional changes, to their versions. We highly recommend not paying any attention to the W3C forks of WHATWG standards.

## 1.13 Differences Between Old HTML and HTML5

In the real world, you'll see a lot of old HTML code. The old code you'll see the most of will probably be XHTML 1.0 because it was the most popular precursor to HTML5. When you see that code, there's no need to be alarmed; today's browsers render XHTML 1.0 code just fine. But in the interest of long-term stability and following your company's coding conventions, you'll sometimes need to convert old HTML code to HTML5. To do so, you need to know the differences between old HTML code and HTML5.

The following differences address the issue of existing web pages that don't match XHTML 1.0's strict syntax. More specifically, the following differences show how HTML5 has loosened up some of its syntax rules as compared to XHTML 1.0:

- With HTML5, there's no longer a requirement to have a quoted value for every attribute. So for some HTML5 attributes, it's legal to include an attribute by itself, without an equals sign or value attached to it. However, standard coding conventions suggest always including the quotation marks.
- With HTML5, there's no longer a requirement to have a / for all void elements. For example, the XHTML specification requires writing the `br` void element with a slash, `<br/>`. The HTML5 specification says you can include or omit the slash. Standard coding conventions suggest always omitting the /.
- With HTML5, there's no longer a requirement to have an end tag for every container. The XHTML specification requires including a `</p>` end tag for every `p` container element. The HTML5 specification says you can include or omit the end tag. Standard coding conventions suggest always including the end tag.

Old versions of HTML (including XHTML 1.0) include some elements that are deemed outdated. In particular, elements whose purpose is to provide formatting are deemed outdated. This is because formatting is supposed to be taken care of by CSS, not HTML elements. To clean things up, such outdated elements are not a part of the HTML5 standard. For example, the `<font>` and

[5] Web Hypertext Application Technology Working Group (WHATWG), "What are the various versions of the HTML spec?" last updated June 14, 2017, https://wiki.whatwg.org/wiki/FAQ#What_are_the_various_versions_of_the_HTML_spec.3F.

`<center>` elements are not supported by HTML5 because they were used to specify font and alignment, which are format-oriented characteristics.

The HTML5 standard includes quite a few new constructs. The following list shows just some of them:

- Structural organization elements—Two examples are the `header` and `footer` elements. We'll cover structural organization elements in Chapter 4.
- Audio and video—The `audio` and `video` elements allow users to play music and video files directly from their browsers without the need of a plug-in. We'll cover `audio` and `video` elements in Chapter 7.
- Canvas—The `canvas` element provides a drawing area and a set of commands that a web programmer can use to draw two-dimensional shapes and animate them. We'll cover the `canvas` element in Chapter 12.
- Drag and drop functionality—The drag and drop constructs provide the ability to drag elements within a web page. This is beyond the scope of this book.
- Web storage functionality—The web storage constructs provide the ability to permanently store data on the browser's computer. This is beyond the scope of this book.
- Geolocation functionality—The geolocation constructs provide the ability to locate the browser's computer. This is beyond the scope of this book.

## 1.14 How to Check Your HTML Code

As described in the previous section, HTML5 contains quite a few differences from earlier HTML standards. So when you're writing your HTML5 code, particularly if you're converting a web page that used an older version of HTML, how do you know if you're following the syntax rules for HTML5? Using this book is a good start, but it doesn't have everything. The formal standards provided by the W3C and WHATWG are comprehensive, but their technical nature and their need to specify details for browser manufacturers make them hard to understand. Here's what we recommend you use for HTML5 reference sites:

https://developers.whatwg.org:
WHATWG's HTML standard for web developers (with browser manufacturer details removed).

https://html.spec.whatwg.org/multipage/semantics.html:
A subset of the WHATWG's standard. It describes all the HTML5 elements.

You'll probably want to refer to one or both of these websites when you have questions about HTML5 syntax.

After you think you've finished creating or modifying a web page, you should check your work by running the W3C's *HTML validation service*, which is formally known as the *Markup Validation Service*. **FIGURE 1.9** shows the validation service's website (https://validator.w3.org). Note the web page's three tabs. With the first tab, Validate by URI, the user enters a web address for the page that is to be checked. For that to work, you need the web page to be uploaded to a web

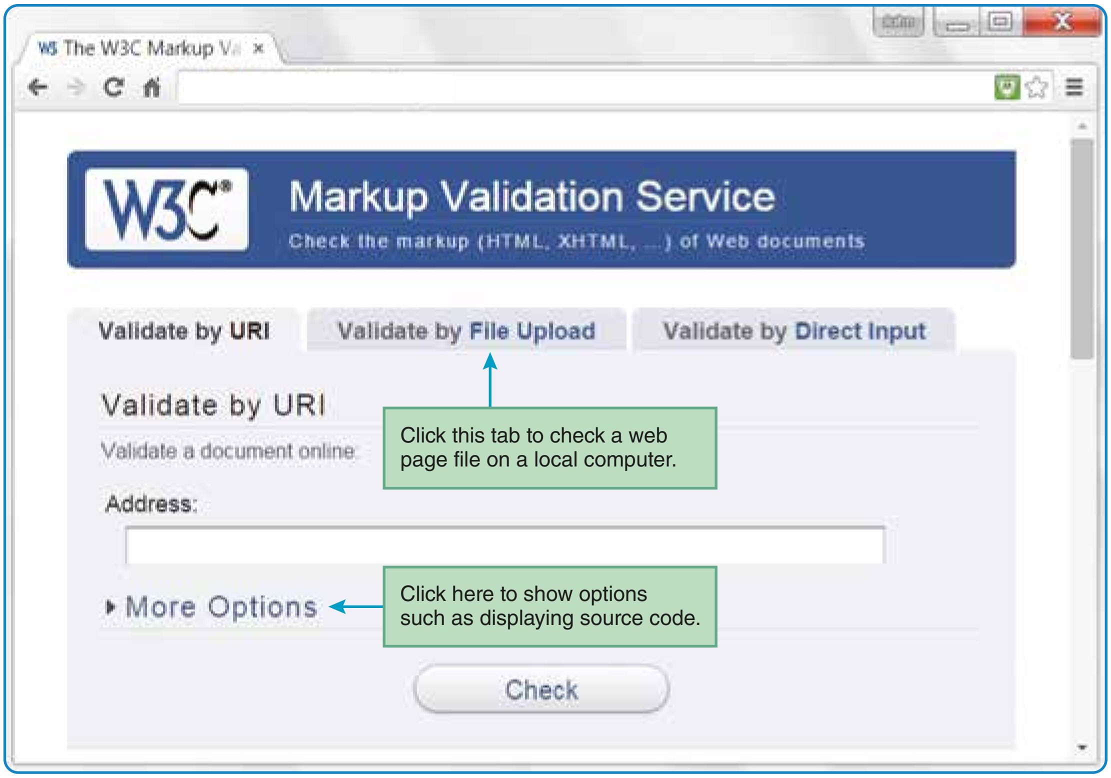

The W3C Markup Va
W3C
Markup Validation Service
Check the markup (HTML, XHTML, ...) of Web documents
Validate by URI
Validate by File Upload
Validate by Direct Input
Validate by URI
Validate a document online:
Address:
More Options
Check
Click this tab to check a web page file on a local computer.
Click here to show options such as displaying source code.

**FIGURE 1.9 W3C's HTML validation service**

server. With the second option, Validate by File Upload, the user selects a file on his or her local computer. With the third option, Validate by Direct Input, the user copies HTML code directly into a large text box. Usually, you'll use the second option, Validate by File Upload, because it's a good idea to test a file stored locally before uploading it. If you upload before testing, you run the risk of having people see your buggy web page.[6]

To get practice with the HTML validation service tool, open your favorite plain text editor or IDE and copy the text from FIGURE 1.10 into it. That text is a modified version of the code in the original Kansas City Weather page. Save the copied code as a file with the name `weatherCheck.html`. Go to the W3C's HTML validation service website and select the Validate by File Upload option. That should generate a Choose File button or a Browse button (different for different browsers). Click the button, navigate to your newly saved `weatherCheck.html` file, and select

[6] A web page that's "buggy" has bugs in it, where a *bug* is an error in a computer program. The term "bug" originated from a malfunctioning program run on one of the early computers, the Harvard Mark II. The computer used mechanical relays to store binary values (e.g., a closed relay was a 1 and an open relay was a 0). As the story goes, a programmer found a squashed moth between the contacts of one of the relays, and the moth's body inhibited the flow of electricity through the relay. After removing the bug, the program worked, and the term "debugging" was born.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>K.C. Weather<title>
</head>

<body>
<h1>Kansas City Weather</h1>
<hr>
<p>
  It should be pleasant today with a high of 95 degrees.<br>
  With a humidity reading of 30%, it should feel like 102 degrees.
<div>
</p>
  Tomorrow's temperatures:<br>
  high 96, low 65
</div>
</body>
</html>
```

**FIGURE 1.10 Source code for modified version of Kansas City Weather web page**

it. Click the More Options box. From the generated options, click the Show Source box. By turning the show source option on, the file's source code will display with line numbers, which can make debugging easier. Now you're ready to let the validation service do its magic. Click the Check button.

The validation service looks for invalid syntax and if it finds any, it prints an error message(s). Here are the error messages the validation service generates for `weatherCheck.html`:

```
Line 20, Column 7: End of file seen when expecting text or an end
tag.
</html>
Line 5, Column 9: Unclosed element title.
  <title>K.C. Weather<title>
```

Unfortunately, error messages can be cryptic, and that is the case for the first of the two error messages. It complains about seeing the end of the file when it's expecting an end tag. Can you figure out what that means? What end tag might be hidden to the validation service? Please do not continue to the next paragraph until after you examine Figure 1.10's code for an end tag problem.

The web page is supposed to display a paragraph first and then a `div` container. To do that, you need start and end tags for a `p` container and then start and end tags for a `div` container. Unfortunately, Figure 1.10's `p` and `div` containers overlap. Specifically, `p`'s end tag comes after `div`'s start tag. That prevents the HTML validation service from seeing `p`'s end tag, which explains the "expecting an end tag" error message.

The second error message, "unclosed element title," is easier to figure out. The `title` element is a container; as such, it needs a start tag and an end tag. Unfortunately, the web page uses

`<title>` for the end tag, instead of `</title>` with a slash. Go to your text editor or IDE and fix the two errors in `weatherCheck.html`. Then run the HTML validation service again and note that the two error messages are gone.

You might have noticed that in creating the modified Kansas City Weather page from the original web page, several things were omitted. The `meta author` and `meta description` elements were omitted, but since they are optional, not required, the HTML validation service doesn't complain. Likewise, the CSS rules were omitted, but since they are optional, not required, the HTML validation service doesn't complain.

## 1.15 CASE STUDY: History of Electric Power

### Preview of Ongoing Case Study

This section begins an extended example of iterative construction of a website. This website's theme is a proposed electric-power "microgrid" for the core of a particular small city—Lawrence, Kansas. A microgrid is a small version of a large electrical power network. In normal operation, it provides valued electrical services to local users and the outside world. If disconnected from the outside world, it employs locally generated solar power and previously stored energy to continue providing critical electrical services to local users.

This ongoing case study generates 10 distinct web pages: Electric Power History, Lawrence Hydropower, Area Description, Microgrid Possibilities, Typical Property, Local Energy, Collector Performance, Electric Power Services, Downtown Properties, and Solar Shadowing. For the most part, each of these web pages is developed completely within the case study section at the end of just one chapter. But in some instances, a web page is iteratively enhanced over two or more chapters. For example, the Electric Power History and Lawrence Hydropower web pages are developed separately at the ends of Chapters 1 and 2, respectively, and then enhanced together at the end of Chapter 3. Also, the Area Description web page is developed at the end of Chapter 3, enhanced at the end of Chapter 7, and enhanced again at the end of Chapter 9.

Each chapter's contribution to the ongoing case study highlights material presented in that chapter. The following is an outline of what each chapter's case study section does:

**Chapter 1**: History of Electric Power
`electricPowerHistory.html` will illustrate HTML structuring (structural elements).

**Chapter 2**: A Local Hydroelectric Power Plant
`lawrenceHydropower.html` will illustrate HTML styling (block and phrasing elements).

**Chapter 3**: Description of a Small City's Core Area
`areaDescription.html` (with `lawrenceMicrogrid.css`) will illustrate CSS styling.

**Chapter 4**: Microgrid Possibilities in a Small City
`microgridPossibilities.html` will illustrate local hypertext navigation.

**Chapter 5**: A Downtown Store's Electrical Generation and Consumption
`typicalProperty.html` will illustrate use of HTML `table` elements.

**Chapter 6**: Local Energy and Home Page with Website Navigation
`localEnergy.html` will illustrate external hypertext navigation and image display.
`index.html` will implement a primitive home page with website navigation.

**Chapter 7**: Using an Image Map for a Small City's Core Area and Website Navigation with a Generic Home Page
An enhancement of Section 3.21's `areaDescription.html` will illustrate mapped images.
A minimal `index.html` will embed other web pages in a home page's `iframe`.

**Chapter 9**: Dynamic Positioning and Collector Performance Web Page
A further enhanced `areaDescription.html` will reposition images as window size changes.
`collectorPerformance.html` will illustrate local JavaScript reading input from a form's `input` and `select` elements and adding rows of data to a `table`.

**Chapter 10**: Collector Performance Details and Nonredundant Website Navigation
JavaScript functions will employ `if` statements, `Math` functions, and `window` properties to compute data for the table in the previous chapter's case study.
JavaScript in an external file will improve the website navigation employed in Chapter 6's case study.

**Chapter 11**: Downtown Properties Data Processing
`properties.html` will illustrate use of arrays and objects to maintain a sorted database.

**Chapter 12**: Solar Shadowing Dynamics
`solarShadowing.html` will illustrate painting of computed geometric shapes on a `canvas`.

## This Chapter's Web Page

The Electric Power History web page presents a brief text description of the history of electric power, followed by a bullet-point list of five important events. It illustrates the following HTML elements: `html`, `head`, `meta`, `title`, `body`, `h2`, `hr`, `p`, `div`, and `br`. Thus, in one example, it provides a simple review of most of the HTML elements introduced in this chapter. **FIGURE 1.11** contains the source code for this web page.

As in Figure 1.4, the `head` element contains three `meta` elements and a `title` element. The first `meta` element has a `charset` attribute, which helps the local browser interpret the web page's binary coding. The next two `meta` elements have `name` and `content` attributes, which help search engines. In the first of these, the `name` attribute gets the value, `author`, and in the second of these, the name attribute gets the value, `description`. The `title` element determines what appears in the tab at the top of the browser's display, and this also helps search engines.

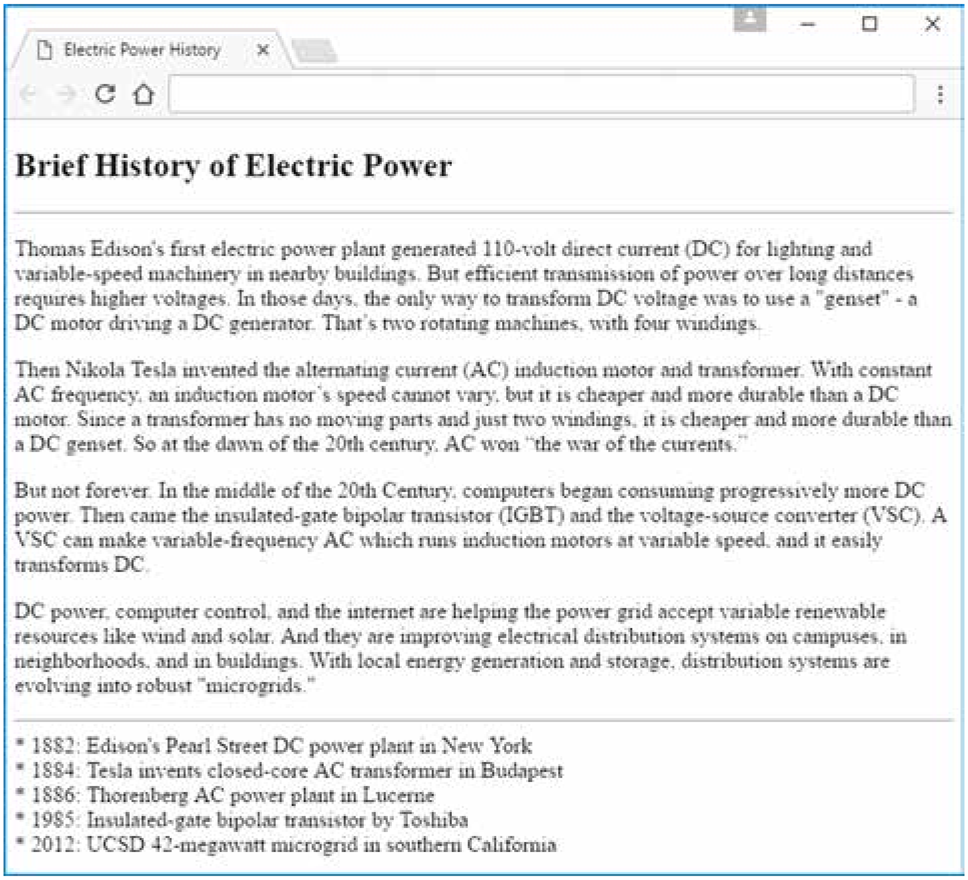
The `body` element contains a heading, a horizontal rule, four text paragraphs, another horizontal rule, and a division with five items, separated by breaks. The selected `h2` heading level avoids overwhelming the page with an excessively large `h1` heading.

In each of the source-code paragraphs in Figure 1.11, text appears as five or six separate lines. Each of these lines begins with two blank spaces and ends with a linefeed. In other words, a

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<meta name="author" content="John Dean">
<title>Electric Power History</title>
</head>

<body>
<h2>Brief History of Electric Power</h2>
<hr>
<p>
  Thomas Edison's first electric power plant generated 110-volt direct
  current (DC) for lighting and variable-speed machinery in nearby
  buildings. But efficient transmission of power over long distances
  requires higher voltages. In those days, the only way to transform DC
  voltage was to use a "genset" - a DC motor driving a DC generator.
  That’s two rotating machines, with four windings.
</p>
<p>
  Then Nikola Tesla invented the alternating current(AC)induction
  motor and transformer. With constant AC frequency, an induction
  motor’s speed cannot vary, but it is cheaper and more durable than a
  DC motor. Since a transformer has no moving parts and just two
  windings, it is cheaper and more durable than a DC genset. So at the
  dawn of the 20th century, AC won “the war of the currents.”
</p>
<p>
  But not forever. In the middle of the 20th Century, computers began
  consuming progressively more DC power. Then came the insulated-gate
  bipolar transistor(IGBT)and the voltage-source converter(VSC). A
  VSC can make variable-ferquency AC which runs induction motors at
  variable speed, and it easily transforms DC.
</p>
<p>
  DC power, computer control, and the internet are helping the power
  grid accept variable renewable resources like wind and solar. And
  they are improving electrical distribution systems on campuses, in
  neighborhoods, and in buildings. With local energy generation and
  storage, distribution systems are evolving into robust "microgrids."
</p>
<hr>
<div>
  * 1882: Edison's Pearl Street DC power plant in New York <br>
  * 1884: Tesla invents closed-core AC transformer in Budapest <br>
  * 1886: Thorenberg AC power plant in Lucerne <br>
  * 1985: Insulated-gate bipolar transistor by Toshiba <br>
  * 2012: UCSD 42-megawatt microgrid in southern California
</div>
</body>
</html>
```

**FIGURE 1.11 Source code for Electric Power History web page**



Brief History of Electric Power

Thomas Edison's first electric power plant generated 110-volt direct current (DC) for lighting and variable-speed machinery in nearby buildings. But efficient transmission of power over long distances requires higher voltages. In those days, the only way to transform DC voltage was to use a "genset" - a DC motor driving a DC generator. That's two rotating machines, with four windings.

Then Nikola Tesla invented the alternating current (AC) induction motor and transformer. With constant AC frequency, an induction motor's speed cannot vary, but it is cheaper and more durable than a DC motor. Since a transformer has no moving parts and just two windings, it is cheaper and more durable than a DC genset. So at the dawn of the 20th century, AC won "the war of the currents."

But not forever. In the middle of the 20th Century, computers began consuming progressively more DC power. Then came the insulated-gate bipolar transistor (IGBT) and the voltage-source converter (VSC). A VSC can make variable-frequency AC which runs induction motors at variable speed, and it easily transforms DC.

DC power, computer control, and the internet are helping the power grid accept variable renewable resources like wind and solar. And they are improving electrical distribution systems on campuses, in neighborhoods, and in buildings. With local energy generation and storage, distribution systems are evolving into robust "microgrids."

* 1882: Edison's Pearl Street DC power plant in New York
* 1884: Tesla invents closed-core AC transformer in Budapest
* 1886: Thorenberg AC power plant in Lucerne
* 1985: Insulated-gate bipolar transistor by Toshiba
* 2012: UCSD 42-megawatt microgrid in southern California

**FIGURE 1.12 Electric Power History web page**

typical text editor would recognize each HTML paragraph as five or six little one-line paragraphs. Distributing a paragraph's text over multiple lines make it easier to read, and each line's initial indentation makes the HTML paragraphing structure easier to recognize.

However, to allow users to resize windows, browsers collapse all contiguous whitespace—including newline characters—into a single-space character. Then they replace one of those single-space characters with a newline character as needed to make running text fit into the current window's width. The browser window in **FIGURE 1.12** shows how this works. Compare the text in Figure 1.12 with that in Figure 1.11, and notice how the browser lengthens and reduces the number of lines as the window's width increases.

Unfortunately, this flexibility comes with a risk. To understand the risk, notice that in Figure 1.11's last text paragraph, at the end of the third line, there is a substantial amount of whitespace. To make your HTML source code more compact, you might be tempted to fill some of this whitespace by hyphenating the word "neighborhoods." Doing that creates more whitespace at the end of the fourth line, and to fill some of this whitespace, you also might be tempted to hyphenate the word "distribution." These changes would make the last text paragraph in Figure 1.11's HTML source code look like this:

```
<p>
  DC power, computer control, and the internet are helping the power grid
  accept variable renewable resources like wind and solar. And they are
  improving electrical distribution systems on campuses, in neighbor-
  hoods, and in buildings. With local energy generation and storage, dis-
  tribution systems are evolving into robust "microgrids."
</p>
```

Then, a browser window whose width is like that in Figure 1.12 would display this modified last text paragraph like this:

DC power, computer control, and the internet are helping the power grid accept
variable renewable resources like wind and solar. And they are improving electrical
distribution systems on campuses, in neighbor- hoods, and in buildings. With
local energy generation and storage, dis- tribution systems are evolving into robust
"microgrids."

Uh oh! Look at the last two lines of the preceding paragraph. Notice that even though the words "neighbor- hoods" and "dis- tribution" are no longer split between two lines, they still contain hyphens. Even worse is that after the hyphen, each word contains that internal space character the browser substitutes for the newline character at the end of each HTML source-code line. Because a browser always substitutes a space character for a newline character, in HTML source code, you should not break even a naturally hyphenated word or term at the end of a line of continuing text.

## Review Questions

### 1.2 Creating a Website

1. What is a web server?
2. Name two features of a web authoring tool.
3. What term is used to describe code that is freely available to view and edit?

### 1.3 Web Page Example

4. What does HTTP stand for?
5. What does URL stand for?

### 1.4 HTML Tags

6. What is the name of the HTML element that generates the smallest heading?

### 1.5 Structural Elements

7. What is syntax?

### 1.6 `title` Element

8. What is the purpose of the `title` element?

### 1.8 HTML Attributes

9. For the `meta` element, what is the purpose of the `name` attribute and the `content` attribute?
10. For the `meta charset` element, what does the value `utf-8` stand for? Specifically, what do "u," "t," "f," and "8" stand for individually?

### 1.10 Cascading Style Sheets Preview

11. What does the following CSS rule do?

```
p {text-align: right;}
```

### 1.11 History of HTML

12. Who designed the HTTP protocol?
13. Who founded the W3C? Name one person.
14. Why did users tend to dislike XHTML 1.1?

### 1.13 Differences Between Old HTML and HTML5

15. The HTML5 standard requires every attribute to have a quoted value. True or false.

## Exercises

1. [after Section 1.2] Search the Internet for a good web hosting service. Describe features of your chosen web hosting service that would make it a good service for your needs (if you don't have any needs, then make some up). You can describe the features in paragraph form or in bullet form, but you must use your own words. Which URL(s) did you use to find your information?
2. [after Section 1.2] Use the statCounter web page to determine the most popular browsers for mobile devices. List the top three browsers in descending order (most popular first).
3. [after Section 1.2] Explain all the parts of a URL. For example, in https://mars.jpl.nasa.gov/msl/images/PIA16082_Mitrofanov1F-thm.jpg, what is each of the following?
   https
   mars
   jpl.nasa.gov
   msl/images
   PIA16082_Mitrofanov1F-thm.jpg
4. [after Section 1.4] What is the difference between a container element and a void element?
5. [after Section 1.5] What is the purpose of the doctype instruction?
6. [after Section 1.5] What is wrong with the following `body` code fragment in terms of the positions of the tags? In your answer, you must (a) use the appropriate term to describe the problem and (b) show how the code should be rearranged.

```
<body>
<p>
  I hate when I'm studying and a velociraptor throws bananas at me.
  Does that happen to anyone else?
  <strong>I hope not!
</p>
</strong>
</body>
```

7. [after Section 1.8] Provide HTML5 code for a paragraph that contains the following Spanish text from Ernesto Sabato's *On Heroes and Tombs*. As always, use proper coding conventions. You must provide code such that search engines are able to identify the paragraph as containing Spanish. You are to assume that only the one paragraph contains Spanish and not the whole page.

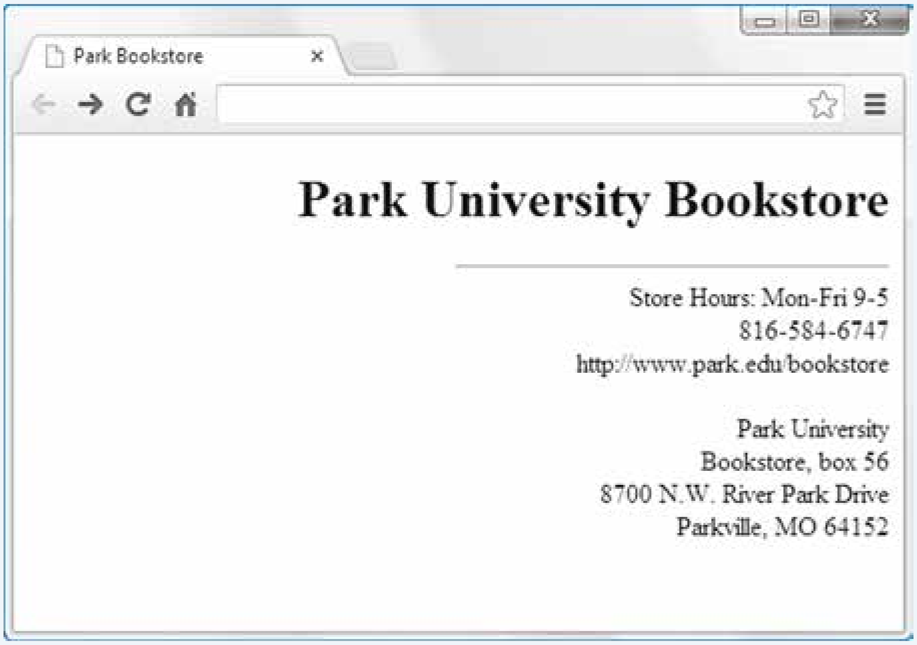
   La vanidad es tan fantastica, que hasta nos induce a preocuparnos de lo que pensaran de nosotros una vez muertos y enterrados.

8. [after Section 1.9] What is the difference between a `p` element and a `div` element?
9. [after Section 1.11] Name something important that Ian Hickson did for the Web, and be specific. Ian is not in this book, so you'll need to look him up.
10. [after Section 1.12] This exercise attempts to get you to explore the W3C's website. Find something interesting on the W3C site. You can describe the features in paragraph form or in bullet form, but you must use your own words. Which URL(s) did you use to find your information?
11. [after Section 1.14] Using the text input mode, enter the following code into W3C's HTML validation service. What error messages are generated? Hint: There are three error messages, but only two problems with the code. You may provide the three error messages verbatim or explain the two problems.

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>test title</title>
</head>
<body>
<h7>test heading
</body>
</html>
```

## Projects

1. Create the following web page, and name the file `parkBookstore.html`. Note that the horizontal line is right aligned. To implement that effect, use the `margin-right` CSS property with a value of `0`.

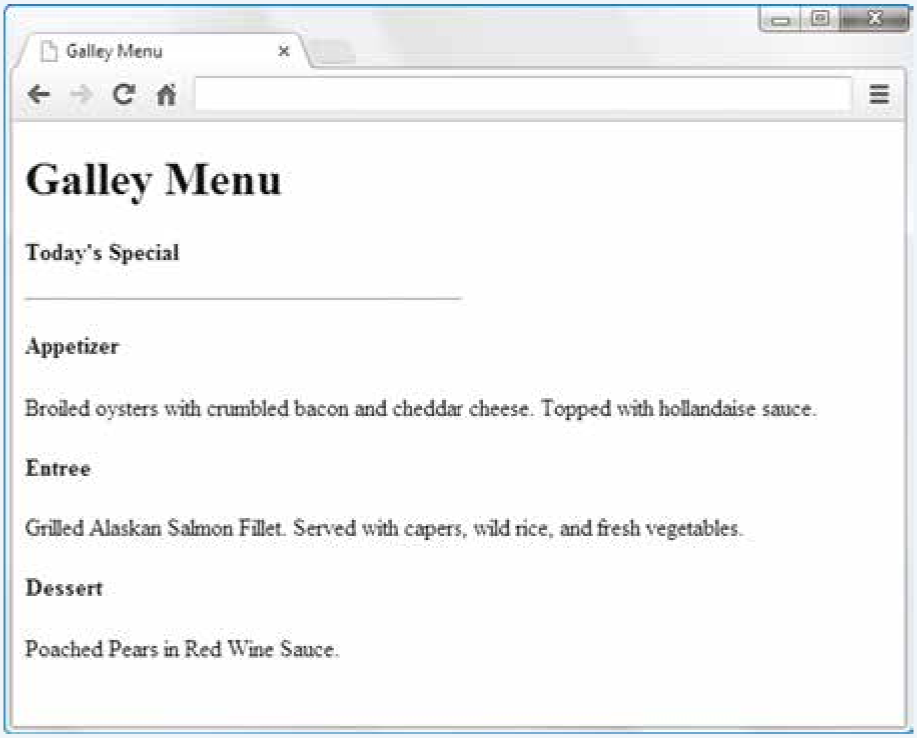


Park University Bookstore

Store Hours: Mon-Fri 9-5
816-584-6747
http://www.park.edu/bookstore

Park University
Bookstore, box 56
8700 N.W. River Park Drive
Parkville, MO 64152

2. Create the following web page, and name the file `galleyMenu.html`. As always, use heading elements for all headings, not just the ones that appear at the top of the page. Note that the horizontal line is left aligned. To implement that effect, use the `margin-left` CSS property with a value of `0`.



# Galley Menu

**Today's Special**

---

**Appetizer**

Broiled oysters with crumbled bacon and cheddar cheese. Topped with hollandaise sauce.

**Entree**

Grilled Alaskan Salmon Fillet. Served with capers, wild rice, and fresh vegetables.

**Dessert**

Poached Pears in Red Wine Sauce.