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

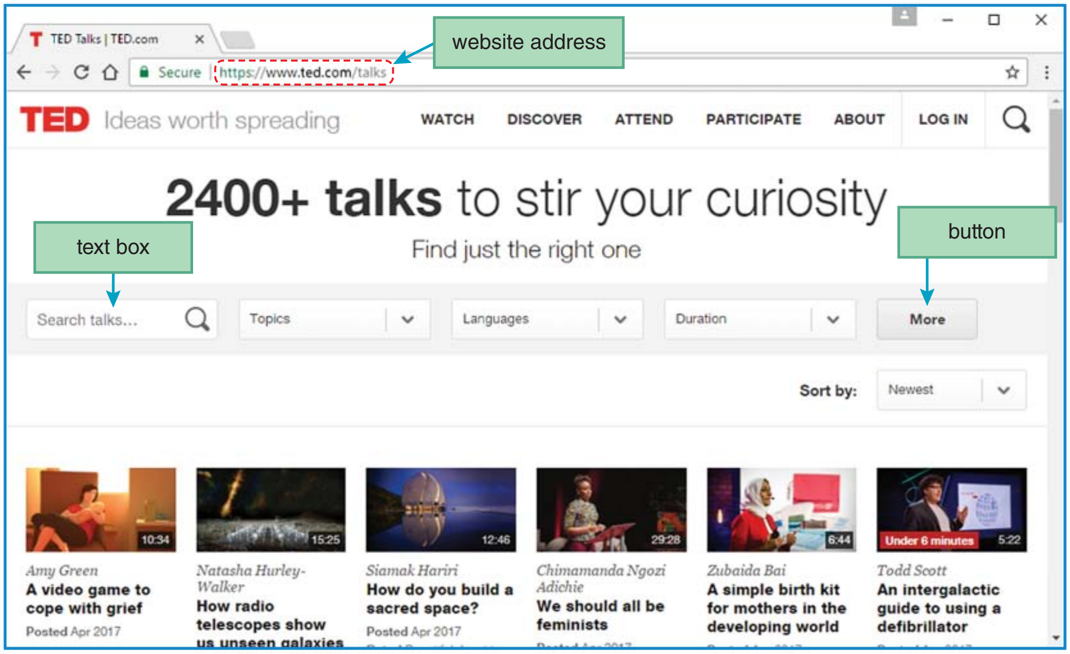

website address
TED Talks | TED.com
https://www.ted.com/talks
TED Ideas worth spreading
WATCH DISCOVER ATTEND PARTICIPATE ABOUT LOG IN
2400+ talks to stir your curiosity
Find just the right one
text box
button
Search talks...
Topics
Languages
Duration
More
Sort by: Newest
Amy Green A video game to cope with grief
Natasha Hurley-Walker How radio telescopes show us unseen galaxies
Siamak Hariri How do you build a sacred space?
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

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/.

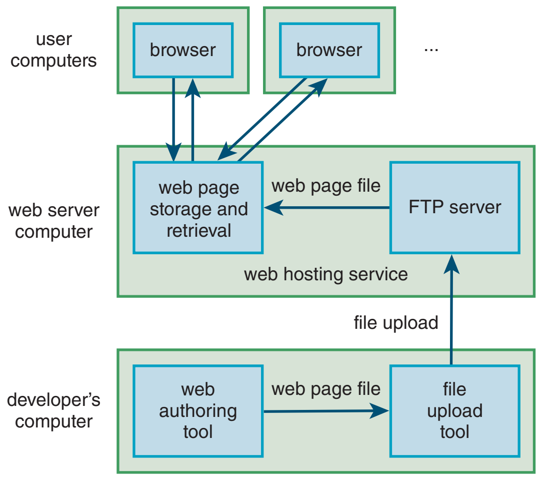


FIGURE 1.2 Website file processing

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.

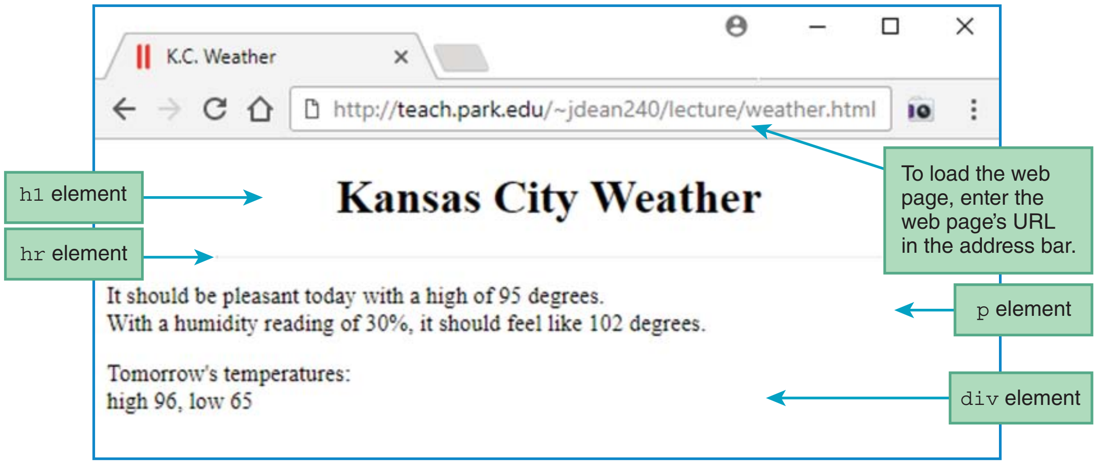
Browsers

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
To load the web page, enter the web page's URL in the address bar.
p element
div element

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 conditions">
<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>
  Tomorrow'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.

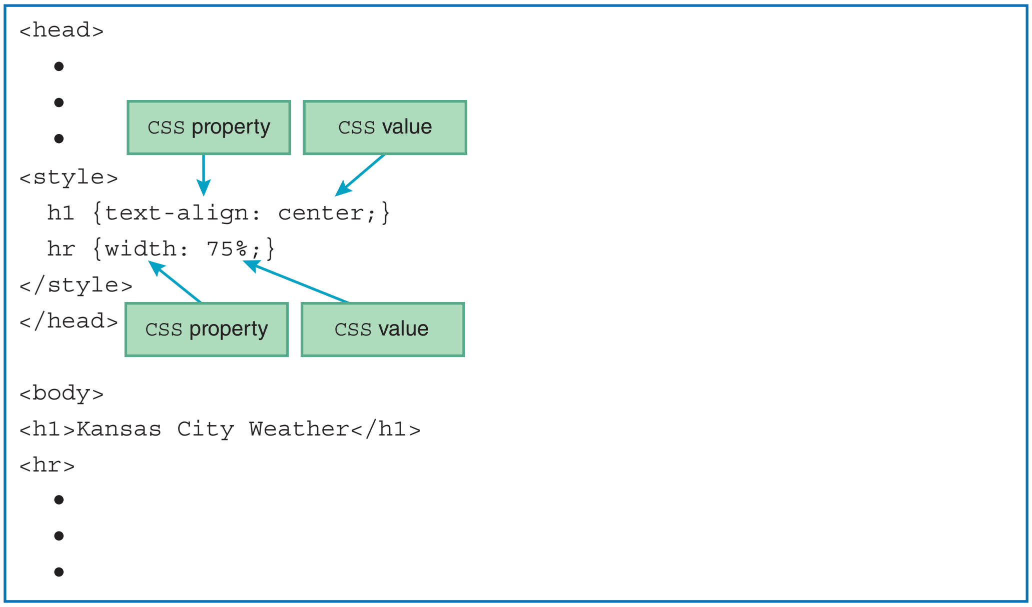
Finally, there's the `br` element, which is used to render a new line. In Figure 1.7, note this line:

```
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