TUTORIAL 1

Getting Started with HTML 5

Creating a Website for a Food Vendor

OBJECTIVES

Session 1.1
- Explore the history of the web
- Create the structure of an HTML document
- Insert HTML elements and attributes
- Insert metadata into a document
- Define a page title

Session 1.2
- Mark page structures with sectioning elements
- Organize page content with grouping elements
- Mark content with text-level elements
- Insert inline images
- Insert symbols based on character codes

Session 1.3
- Mark content using lists
- Create a navigation list
- Link to files within a website with hypertext links
- Link to email addresses and telephone numbers

Case | *Curbside Thai*

Sajja Adulet is the owner and master chef of Curbside Thai, a restaurant owner and now food truck vendor in Charlotte, North Carolina that specializes in Thai dishes. Sajja has hired you to develop the company's website. The website will display information about Curbside Thai, including the truck's daily locations, menu, catering opportunities, and contact information. Sajja wants the pages to convey the message that customers will get the same great food and service whether they order in the restaurant or from the food truck. Some of the materials for these pages have already been completed by a former employee and Sajja needs you to finish the job by converting that work into a collection of web page documents. To complete this task, you'll learn how to write and edit HTML 5 code and how to get your HTML files ready for display on the World Wide Web.

STARTING DATA FILES

tutorial
ct_catering_txt.html
ct_contact_txt.html
ct_locations_txt.html
ct_menu_txt.html
ct_reviews._txt.html + 16 files

review
mp_catering_txt.html
mp_events_txt.html
mp_index_txt.html
mp_menu_txt.html + 5 files

code1
code1-1_txt.html

code2
code1-2_txt.html

code3
code1-3_txt.html + 7 files

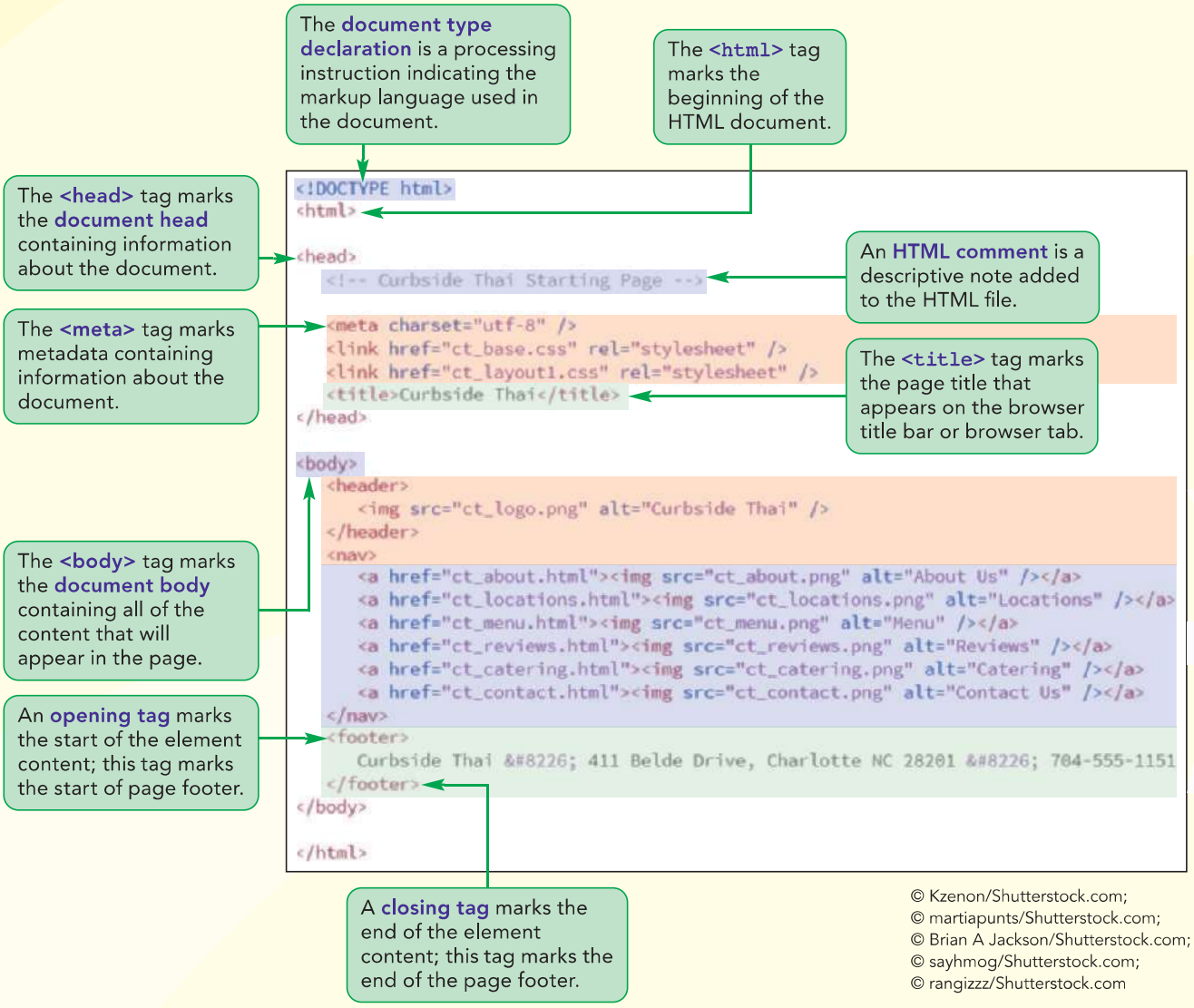
code4
code1-4_txt.html + 2 files

case1
jtc_index_txt.html
jtc_services_txt.html
+ 6 files

case2
dr_faq_txt.html
dr_index_txt.html
dr_info_txt.html + 9 files

demo
demo_characters.html
demo_html.html
+ 3 files

Session 1.1 Visual Overview:



The Structure of an HTML Document

Kzenon/Shutterstock.com; © Courtesy Patrick Carey; martiapunts/Shutterstock.com; Brian A Jackson/Shutterstock.com; A Studios/Shutterstock.com; rangizzz/Shutterstock.com

Exploring the World Wide Web

It is no exaggeration to say that the World Wide Web has had as profound an effect on human communication as the printing press. One key difference is that operation of the printing press was limited to a few select tradesmen but on the web everyone can be a publisher of a website. Before creating your first website, you'll examine a short history of the web because that history impacts the way you write code for your web pages. You'll start by exploring the basic terminology of computer networks.

Networks

A **network** is a structure in which information and services are shared among devices known as **nodes** or **hosts**. A host can be any device that is capable of sending or receiving data electronically. The most common hosts that you will work with are desktop computers, laptops, tablets, mobile phones, and printers.

A host that provides information or a service to other devices on the network is called a **server**. For example, a print server provides printing services; a file server provides storage space for saving and retrieving files. The device receiving these services is called a **client**. A common network design is the **client-server network**, in which the clients access information provided by one or more servers.

Networks are classified based on the range of devices they cover. A network confined to a small geographic area, such as within a building or department, is referred to as a **local area network** or **LAN**. A network that covers a wider area, such as several buildings or cities, is called a **wide area network** or **WAN**. Wide area networks typically consist of two or more interconnected local area networks. The largest WAN in existence is the **Internet**, which incorporates an almost uncountable number of networks and hosts involving computers, mobile devices (such as phones, tablets, and so forth), MP3 players, and gaming systems.

Locating Information on a Network

The biggest obstacle to effectively using the Internet is the network's sheer scope and size. Most of the early Internet tools required users to master a bewildering array of terms, acronyms, and commands. Because network users had to be well versed in computers and network technology, Internet use was largely limited to programmers and computer specialists working for universities, large businesses, and the government.

The solution to this problem was developed in 1989 by Timothy Berners-Lee and other researchers at the CERN nuclear research facility near Geneva, Switzerland. They needed an information system that would make it easy for their researchers to locate and share data on the CERN network, and so developed a system of hypertext documents. **Hypertext** is a method of organization in which data sources are interconnected through a series of links or **hyperlinks** activated to jump from one data source to another. Hypertext is ideally suited for the Internet because end users don't need to know where a service is located—they only need to know how to activate the link. The effectiveness of this technique quickly spread beyond Geneva and was adopted across the Internet. The totality of these interconnected hypertext documents became known as the **World Wide Web**. The fact that the Internet and the World Wide Web are synonymous in many users' minds is a testament to the success of the hypertext approach.

Web Pages and Web Servers

Documents on the web are stored on **web servers** in the form of **web pages** and accessed through a software program called a **web browser**. The browser retrieves the

document from the web server and renders it in a form readable on a client device. However, because there is a wide selection of client devices ranging from desktop computers to mobile phones to screen readers that relay data aurally, each web page must be written in code that is compatible with every device. How does the same document work with so many different devices? To understand, you need to look at how web pages are created.

Introducing HTML

A web page is a simple text file written in **HTML** (**Hypertext Markup Language**). You've already read about hypertext, but what is a markup language? A **markup language** is a language that describes the content and structure of a document by "marking up" or tagging, different document elements. For example, this tutorial contains several document elements such as the tutorial title, main headings, subheadings, paragraphs, figures, figure captions, and so forth. Using a markup language, each of these elements could be tagged as a distinct item within the "tutorial document." Thus, a Hypertext Markup Language is a language that supports tagging distinct document elements and connecting documents through hypertext links.

The History of HTML

In the early years, no single organization defined the rules or **syntax** of HTML. Browser developers were free to define and modify the language in different ways that, of course, led to problems as different browsers supported different "flavors" of HTML and a web page that was written based on one browser's standard might appear totally different when rendered by another browser. Ultimately, a group of web designers and programmers called the **World Wide Web Consortium**, or the **W3C**, settled on a set of standards or specifications for all browser manufacturers to follow. The W3C has no enforcement power, but, because using a uniform language is in everyone's best interest, the W3C's recommendations are usually followed, though not always immediately. Each new version of HTML goes through years of discussion and testing before it is formally adopted as the accepted standard. For more information on the W3C and its services, see its website at *www.w3.org*.

By 1999, HTML had progressed to the fourth version of the language, **HTML 4.01**, which provided support for multimedia, online commerce, and interactive scripts running within the web page. However, there were still many incompatibilities in how HTML was implemented across different browsers and how HTML code was written by web developers. The W3C sought to take control of what had been a haphazard process and enforce a stricter set of standards in a different version of the language called **XHTML** (**Extensible Hypertext Markup Language**). By 2002, the W3C had released the specifications for XHTML 1.1. But XHTML 1.1 was intended to be only a minor upgrade on the way to XHTML 2.0, which would correct many of the deficiencies found in HTML 4.01 and become the future language of the web. One problem was that XHTML 2.0 would not be backward compatible with HTML and, as a result, older websites could not be easily brought into the new standard.

Web designers rebelled at this development and, in response, the **Web Hypertext Application Technology Working Group** (**WHATWG**) was formed in 2004 with the mission to develop a rival version to XHTML 2.0, called **HTML 5**. Unlike XHTML 2.0, HTML 5 would be compatible with earlier versions of HTML and would not apply the same strict standards that XHTML demanded. For several years, it was unclear which specification would win out; but by 2006, work on XHTML 2.0 had completely stalled and the W3C issued a new charter for WHATWG to develop HTML 5 as the de facto standard for the next generation of HTML. You can learn more about WHATWG and its current projects at *www.whatwg.org*. The current version of HTML is HTML 5.2, which achieved Recommendation status in 2017.

TIP

You can find out which browsers support the features of HTML 5 by going to the website *caniuse.com*.

As HTML has evolved, features and code found in earlier versions of the language are often **deprecated**, or phased out, and while deprecated features might not be part of HTML 5, that doesn't mean that you won't encounter them in your work—indeed, if you are maintaining older websites, you will often need to interpret code from earlier versions of HTML. Moreover, there are still many older browsers and devices in active use that do not support HTML 5. Thus, a major challenge for website designers is writing code that takes advantage of HTML 5 but is still accessible to older technology.

Figure 1–1 summarizes some of the different versions of HTML that have been implemented over the years. You can read detailed specifications for these versions at the W3C website.

Figure 1–1 HTML version history

Version	Date	Description
HTML 1.0	1989	The first public version of HTML
HTML 2.0	1995	HTML version that added interactive elements including web forms
HTML 3.2	1997	HTML version that provided additional support for web tables and expanded the options for interactive form elements and a scripting language
HTML 4.01	1999	HTML version that added support for style sheets to give web designers greater control over page layout and appearance, and provided support for multimedia elements such as audio and video
XHTML 1.0	2001	A reformulation of HTML 4.01 using the XML markup language in order to provide enforceable standards for HTML content and to allow HTML to interact with other XML languages
XHTML 2.0	discontinued in 2009	The follow-up version to XHTML 1.1 designed to fix some of the problems inherent in HTML 4.01 syntax
HTML 5.0	2012	HTML version providing support for mobile design, semantic page elements, column layout, form validation, offline storage, and enhanced multimedia
HTML 5.2	2017	The current version of HTML 5

This book focuses on HTML 5, but you will also review some of the specifications for HTML 4.01 and XHTML 1.1. Deprecated features from older versions of HTML will be noted as such in the text.

Tools for Working with HTML

Because HTML documents are simple text files, the first tool you will need is a text editor. You can use a basic text editor such as Windows Notepad or TextEdit for the Macintosh, but it is highly recommended that you use one of the many inexpensive editors that provide built-in support for HTML. These editors include syntax checking to weed out errors and automatic insertion of HTML code. Some of the more popular HTML editors are Notepad++ (*notepad-plus-plus.org*), Eclipse (*www.eclipse.org*), and CoffeeCup (*www.coffeecup.com*).

These enhanced editors are a good way to start learning HTML and they will be all you need for most basic projects, but professional web developers working on large websites will quickly gravitate toward using a web **IDE** (**Integrated Development Environment**), which is a software package providing comprehensive coverage of all

phases of the development process from writing HTML code to creating scripts for programs running on web servers. Some of the popular IDEs for web development include Adobe Dreamweaver (*www.adobe.com*), Aptana Studio (*www.aptana.com*), NetBeans IDE (*netbeans.org*), and Komodo IDE (*komodoide.com*). Web IDEs can be very expensive, but most software companies will provide a free evaluation period for you to test their product to see if it meets your needs.

Content Management Systems and Frameworks

You can also invest in a **web content management system** (**wcms**) which provides authoring tools for website content and administration. Management systems provide prepackaged templates so that users can get websites up and running with only a minimal knowledge of HTML. Popular content management systems include WordPress (*www.wordpress.org*), Joomla (*www.joomla.org*), and Drupal (*www.drupal.org*). Content management systems are not without drawbacks. A wcms can be expensive to maintain and put extra load on server resources. In addition, the templates and authoring tools can be difficult to modify if they don't exactly meet your needs.

A website usually involves the integration of many technologies and languages beyond HTML, including databases for storing and retrieving data and programs running on the web server for managing electronic commerce and communication. Managing all those technologies is the job of a **web framework** that provides the foundation of the design and deployment of web applications. Popular frameworks include Ruby on Rails (*rubyonrails.org*), ASP.NET (*www.asp.net*), AngularJS (*angularjs.org*), and Django (*www.djangoproject.com*).

Choosing among all these tools might seem intimidating to you. The bottom line is that no matter what tools you use, the final code for the website is written in HTML. So, even if that code is generated by a framework or content management system, you still need to understand HTML to effectively manage your website. In this book, we'll try to keep things as simple as possible: just you, a text editor, and a web browser creating a foundation for future study.

Testing your Code

TIP

You can analyze each browser for its compatibility with HTML 5 at the website *www.html5test.com*.

Once you've written your code, you can test whether your HTML code employs proper syntax and structure by validating it at the W3C validation website (*validator.w3.org*). **Validators**, like the one available through the W3C website, are programs that test code to ensure that it contains no syntax errors. The W3C validator will highlight all of the syntax errors in your document with suggestions about how to fix those errors.

Finally, you'll need to test it to ensure that your content is rendered correctly. You should test your code under a variety of screen resolutions, on several different browsers and, if possible, on different versions of the same browser because users are not always quick to upgrade their browsers. What may look good on a widescreen monitor might look horrible on a mobile phone. At a minimum you should test your website using the following popular browsers: Google Chrome, Internet Explorer, Apple Safari, Mozilla Firefox, and Opera.

It is not always possible to load multiple versions of the same browser on one computer, so, in order to test a website against multiple browser versions, professional designers will upload their code to online testing services that report on the website's compatibility across a wide range of browsers, screen resolutions, and devices, including both desktop and mobile devices. Among the popular testing services are BrowserStack (*www.browserstack.com*), CrossBrowserTesting (*www.crossbrowsertesting.com*), and Browsera (*www.browsera.com*). Most of these sites charge a monthly connection fee with a limited number of testing minutes, so you should not upload your code until you are past the initial stages of development.

Exploring an HTML Document

Now that you have reviewed the history of the web and some of the challenges in developing your own website, you will look at the code of an actual HTML file. To get you started, Sajja Adulet has provided you with the ct_start.html file containing the code for the initial page users see when they access the Curbside Thai website. Open Sajja's file now.

TIP

All HTML files have the file extension .html or .htm.

To open the ct_start.html file:

1. Use the editor of your choice to open the **ct_start.html** file from the html01 ▸ tutorial folder.

   Figure 1–2 shows the complete contents of the file as viewed in the Notepad++ editor.

Figure 1–2 **Elements and attributes from an HTML document**

Trouble? Depending on your editor and its configuration, the text style applied to your code might not match that shown in Figure 1–2. This is not a problem. Because HTML documents are simple text files, any text styles are a feature of the editor and have no impact on how the document is rendered by the browser.

2. Scroll through the document to become familiar with its content but do not make any changes to the text.

The Document Type Declaration

The first line in an HTML file is the document type declaration or doctype, which is a processing instruction indicating the markup language used in the document. The browser uses the document type declaration to know which standard to use for displaying the content. For HTML 5, the doctype is entered as

```
<!DOCTYPE html>
```

You might also see the doctype entered in lowercase letters as

```
<!doctype html>
```

Both are accepted by all browsers. Older versions of HTML had more complicated doctypes. For example, the doctype for HTML 4.01 is the rather foreboding

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
```

You might even come across older HTML files that do not have a doctype. Because early versions of HTML did not require a doctype, many browsers interpret the absence of the doctype as a signal that the page should be rendered in **quirks mode**, based on styles and practices from the 1990s and early 2000s. When the doctype is present, browsers will render the page in **standards mode**, employing the most current specifications of HTML. The difference between quirks mode and standards mode can mean the difference between a nicely laid-out page and a confusing mess, so always put your HTML 5 file in standards mode by including the doctype.

Introducing Element Tags

The fundamental building block in every HTML document is the **element tag**, which marks an element in the document. A **starting tag** indicates the beginning of that element, while an **ending tag** indicates the ending. The general syntax of a two-sided element tag is

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

where *element* is the name of the element, *content* is the element's content, `<element>` is the starting tag, and `</element>` is the ending tag. The following code marks a paragraph element enclosed within the `<p>` and `</p>` tags:

```
<p>Welcome to Curbside Thai.</p>
```

The `<p></p>` tags indicate the presence of a paragraph and the text *Welcome to Curbside Thai*. comprises the paragraph text.

Not every element tag encloses document content. **Empty elements** are elements that are either nontextual (such as images) or contain directives to the browser about how the page should be treated. An empty element is entered using one of the following forms of the **one-sided element tag**:

```
<element />
```

or

```
<element>
```

The following `br` element indicates the presence of a line break in the text, and thus does not contain any content:

```
<br />
```

Note that, while this code could also be entered as `<br>`, the ending slash `/>` form is required in XHTML documents as well as other markup languages. While HTML 5 allows for either form, it's a good idea to get accustomed to using the ending slash `/>` form if you intend to work with other markup languages. We'll follow the `/>` convention in the code in this book.

Elements can contain other elements, which are called **nested elements**. In the following code, the `em` element (used to mark emphasized text) is nested within the paragraph element by placing the `<em>` tag completely within the `<p>` tag.

Proper syntax:

```
<p>Welcome to <em>Curbside Thai</em>.</p>
```

When nesting one element inside of another, the entire code of the inner element must be contained within the outer element, including opening and closing tags. Thus, it would not be correct to place the closing tag for the `em` element outside of the `p` element as in the following code:

Improper syntax:

```
<p>Welcome to <em>Curbside Thai</p>.</em>
```

Now that you've examined the basics of tags, you'll examine at how they're organized within an HTML file.

The Element Hierarchy

The entire structure of an HTML document can be thought of as a set of nested elements in a hierarchical tree. At the top of the tree is the `html` element marking the entire document. Within the `html` element is the `head` element enclosing information about the document itself and the `body` element enclosing the content of the web page. Thus, the general structure of an HTML file, like the one shown in Figure 1–2, is

```
<!DOCTYPE html>
<html>
<head>
   head content
</head>

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

where *head content* and *body content* are nested elements placed within the document head and body. Note that the `body` element is always placed after the `head` element.

REFERENCE

Creating the Basic Structure of an HTML File

- To create the basic structure of an HTML file, enter the tags

```
<!DOCTYPE html>
<html>
<head>
   head content
</head>

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

where *head*, *content*, and *body content* contain nested elements that mark the content of the head and body sections.

Introducing Element Attributes

TIP

Attributes can be listed in any order but they must come after the element name and be separated from each other by a blank space; each attribute value must be enclosed within single or double quotation marks.

Elements often contain one or more **element attributes** providing additional information about the purpose of the element or how the element should be handled by the browser. The general syntax of an element attribute within a two-sided tag is

```
<element attr1="value1" attr2="value2" …>
   content
</element>
```

Or, for a one-sided tag

```
<element attr1="value1" attr2="value2" … />
```

where *attr1*, *attr2*, and so forth are attributes associated with *element* and *value1*, *value2*, and so forth are the corresponding attribute values. For example, the following code adds the `id` attribute with the value "intro" to the `<p>` tag in order to identify the paragraph as an introductory paragraph.

```
<p id="intro">Welcome to Curbside Thai.</p>
```

Each element has its own set of attributes but, in addition to these element-specific attributes, there is a core set of attributes that can be applied to almost every HTML element. Figure 1–3 lists some of the most commonly used core attributes; others are listed in Appendix B.

Figure 1–3 Commonly used core HTML attributes

Attribute	Description
`class="text"`	Defines the general classification of the element
`dir="ltr|rtl|auto"`	Defines the text direction of the element content as left-to-right, right-to-left, or determined by the browser
`hidden`	Indicates that the element should be hidden or is no longer relevant
`id="text"`	Provides a unique identifier for the element
`lang="text"`	Specifies the language of the element content
`style="definition"`	Defines the style or appearance of the element content
`tabindex="integer"`	Specifies the tab order of the element (when the tab button is used to navigate the page)
`title="text"`	Assigns a title to the element content

For attributes that do not require a value, HTML supports **attribute minimization** by removing the attribute value completely. For example, the `hidden` attribute used in the following code does not require a value; its mere presence indicates that the marked paragraph should be hidden in the rendered page.

```
<p hidden>Placeholder Text</p>
```

Attribute minimization is another example of how HTML 5 differs from other markup languages such as XHTML in which minimization is not allowed and all attributes must have attribute values.

REFERENCE

Adding an Attribute to an Element

- To add an attribute to an element, enter

```
<element attr1="value1" attr2="value2" …>
   content
</element>
```

where *attr1*, *attr2*, and so forth are HTML attributes associated with *element* and *value1*, *value2*, and so forth are the corresponding attribute values.

Handling White Space

An HTML file is composed only of text characters and white-space characters. A **white-space character** is any empty or blank character such as a space, tab, or line break. The browser reading the HTML code ignores the presence of white-space characters between element tags and makes no distinction between spaces, tabs, or line breaks. Thus, a browser will treat the following two pieces of code the same:

```
<p>Welcome to <em>Curbside Thai</em>.</p>
```

and

```
<p>
   Welcome to <em>Curbside Thai</em>.
</p>
```

The browser also collapses consecutive occurrences of white-space characters into a single occurrence, so that the text of the paragraph in the following code is still treated as "Welcome to Curbside Thai", ignoring the extra white spaces between "Curbside" and "Thai".

```
<p>
   Welcome to <em>Curbside      Thai</em>.
</p>
```

The bottom line is that it doesn't matter how you lay out your HTML code because the browser is only interested in the text content and not how that text is entered. This means you can make your file easier to read by indenting lines and by adding extra white-space characters to separate one code block from another. However, this also means that any formatting you do for the page text to make the code more readable, such as tabs or extra white spaces, is *not* transferred to the web page.

Viewing an HTML File in a Browser

The structure of the HTML file shown in Figure 1–2 should now be a little clearer, even if you don't yet know how to interpret the meaning and purpose of each of element and attribute. To see what this page looks like, open it within a web browser.

To open the ct_start.html file in a web browser:

1. Open your web browser. You do not need to be connected to the Internet to view local files stored on your computer.
2. After your browser loads its home page, open the ct_start.html file from the html01 ▸ tutorial folder. Figure 1–4 shows the page as it appears on a mobile phone and on a tablet device. The two devices have different screen widths, which affects how the page is rendered.

Figure 1–4 The Curbside Thai starting page as rendered by a mobile and tablet device

mobile device tablet device

Trouble? If you're not sure how to open a local file with your browser, check for an Open or Open File command under the browser's File menu. You can also open a file by double-clicking the file name from within Windows Explorer or Apple Finder.

3. Reduce the width of your browser window and note that when the width falls below a certain value (in this case 480 pixels), the layout automatically changes to a stacked row of images (as shown in the mobile device image in Figure 1–4) that are better suited to the narrower layout.
4. Increase the width of the browser window and confirm that the layout changes to a 2×3 grid of images (as shown in the tablet device image in Figure 1–4), which is a design more appropriate for the wider window.

Figure 1–4 illustrates an important principle: *HTML does not describe the document's appearance, it only describes the document's content and structure*. The same HTML document can be rendered differently on different devices or screen sizes. The actual appearance of the document is determined by style sheets—a topic you'll explore later in this tutorial.

Creating an HTML File

Now that you've studied the structure of an HTML file, you'll start creating your own documents for the Curbside Thai website. Sajja wants you to create a web page containing information about the restaurant. Start by inserting the doctype and the markup tags for the `html`, `head`, and `body` elements. You will also specify English (`en`) as the language of the web page by adding the `lang` attribute to the `<html>` tag.

To begin writing the HTML file:

TIP

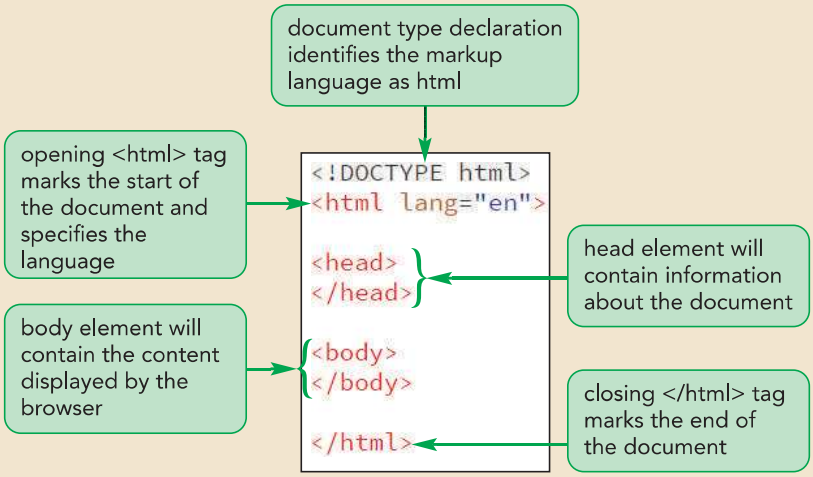
HTML filenames should be entered in lowercase letters and have *no* blank spaces.

1. Using the editor of your choice, create a new blank HTML file in the html01 ▸ tutorial folder, saving the file as **ct_about.html**.
2. Enter the following code into the file:

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

<head>
</head>

<body>
</body>

</html>
```

Figure 1–5 shows the initial elements in the document.

Figure 1–5 **Initial structure of the ct_about.html file**



3. Save your changes to the file.

Next, you'll add elements to the document head.

PROSKILLS

Written Communication: Writing Effective HTML Code

Part of writing good HTML code is being aware of the requirements of various browsers and devices, as well as understanding the different versions of the language. Here are a few guidelines for writing good HTML code:

- Become well versed in the history of HTML and the various versions of HTML and XHTML. Unlike other languages, HTML's history does impact how you write your code.
- Know your market. Do you have to support older browsers, or have your clients standardized on one particular browser or browser version? Will your web pages be viewed on a single device such as a computer, or do you have to support a variety of devices?
- Test your code on several different browsers and browser versions. Don't assume that if your page works in one browser, it will work in other browsers or even in earlier versions of the same browser. Also check on the speed of the connection. A large file that performs well with a high-speed connection might be unusable with a slower connection.
- Read the documentation on the different versions of HTML and XHTML at the W3C website and keep up to date with the latest developments in the language.

To effectively communicate with customers and users, you need to make sure your website content is always readable. Writing good HTML code is a great place to start.

Creating the Document Head

The document head contains **metadata**, which is content that describes the document or provides information about how the document should be processed by the browser. Figure 1–6 describes the different metadata elements found in the document head.

Figure 1–6 HTML metadata elements

Element	Description
`head`	Contains a collection of metadata elements that describe the document or provide instructions to the browser
`base`	Specifies the document's location for use with resolving relative hypertext links
`link`	Specifies an external resource that the document is connected to
`meta`	Provides a generic list of metadata values such as search keywords, viewport properties, and the file's character encoding
`script`	Provides programming code for programs to be run within the document
`style`	Defines the display styles used to render the document content
`title`	Stores the document's title or name, usually displayed in the browser title bar or on a browser tab

The first metadata you'll add to the About Curbside Thai web page is the `title` element.

Setting the Page Title

The `title` element is part of the document head because it's not displayed within the web page, but rather in the browser title bar or browser tab. Page titles are defined using the following `title` element

```
<title>document title</title>
```

where *document title* is the text of the title. Add a page title to the Curbside Thai page now.

REFERENCE

Adding a Document Title

- To define the document title, enter the following tag into the document head:

```
<title>document title</title>
```

where *document title* is the text that will appear on the browser title bar or a browser tab.

To insert the document title:

TIP

Document titles should be no more than 64 characters in length to ensure that the text fits on the browser title bar or a browser tab.

1. Directly after the opening `<head>` tag, insert the following `title` element, indented to make the code easier to read.

```
<title>About Curbside Thai</title>
```

Figure 1–7 highlights the code for the page title.

Figure 1–7 Entering the document title

title text that appears in the browser title bar or on a browser tab

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

<head>
   <title>About Curbside Thai</title>
</head>

<body>
</body>

</html>
```

2. Save your changes to the file.

Adding Metadata to the Document

Another metadata is the `meta` element, which is used for general lists of metadata values. The `meta` element structure is

```
<meta attributes />
```

where *attributes* define the type of metadata that is to be added to a document. Figure 1–8 lists the attributes of the `meta` element.

Figure 1–8 Attributes of the meta element

Attribute	Description
`charset="encoding"`	Specifies the character encoding used in the HTML document
`content="text"`	Provides the value associated with the `http-equiv` or `name` attributes
`http-equiv="content-type|default-style|refresh"`	Provides an HTTP header for the document's content, default style, or refresh interval (in seconds)
`name="text"`	Sets the name associated with the metadata

For example, you can use the following `meta` element to provide a collection of keywords for the Curbside Thai website that would aid web search engines, such as Google or Bing search tools, to locate the page for potential customers:

```
<meta name="keywords" content="Thai, restaurant, Charlotte, food" />
```

The `name` attribute defines the type of metadata and the content attribute provides the data values. HTML does not specify a set of values for the `name` attribute, but commonly used names include `keywords`, `description`, `author`, and `viewport`.

TIP

The title element and the charset meta element are both required in a valid HTML 5 document.

Another use of the `meta` element is to define the character encoding used in the HTML file. **Character encoding** is the process by which the computer converts text into a sequence of bytes when it stores the text and then converts those bytes back into characters when the text is read. The most common character encoding is **UTF-8**, which supports almost all of the characters you will need. To indicate that the document is written using UTF-8, add the following `meta` element to the document head:

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

The `charset` attribute was introduced in HTML 5 and replaces the following more complicated expression used in earlier versions of HTML:

```
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8" />
```

REFERENCE

Adding Metadata to the Document

- To define the character encoding used in the document, enter

  `<meta charset="encoding" />`

  where *encoding* is the character encoding used in the document.
- To define search keywords associated with the document, enter

  `<meta name="keywords" content="terms" />`

  where *terms* is a comma-separated list of keyword terms.

Add `meta` elements to the document head now, providing the character set and a list of keywords describing the page.

To insert metadata:

TIP

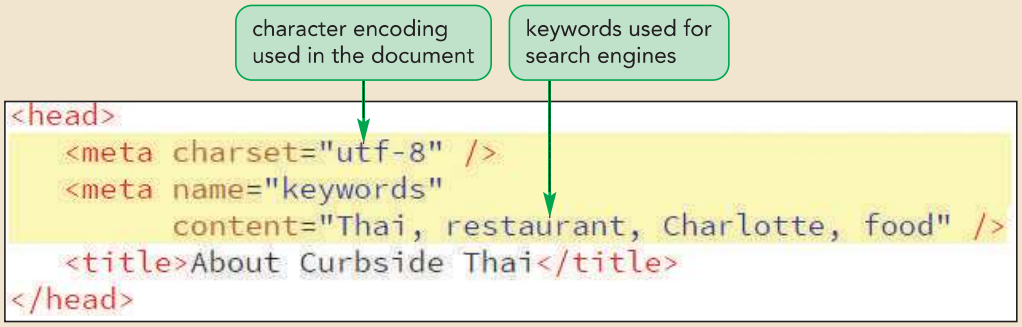
The <meta> tag that defines the character encoding should always be the first meta element in the document head.

1. Directly after the opening <head> tag, insert the following meta elements, indented to make the code easier to read:

```
<meta charset="utf-8" />
<meta name="keywords"
  content="Thai, restaurant, Charlotte, food" />
```

Figure 1–9 highlights the newly added meta elements used in the document head.

Figure 1–9 Adding metadata to a document



2. Save your changes to the file.
3. Open the **ct_about.html** file in your browser. Confirm that the browser tab or browser title bar contains the text "About Curbside Thai". There should be no text displayed in the browser window because you have not added any content to the page body yet.

Before continuing with your edits to the ct_about.html file, you should document your work. You can do this with HTML comments.

Adding Comments to Your Document

An HTML comment is descriptive text that is added to the HTML file but that does not appear in the page. Comments can include the name of the document's author, the date the document was created, and the purpose for which the document was created. Comments are added with the following markup:

```
<!-- comment -->
```

where *comment* is the text of the comment or note. For example, the following code inserts a comment describing the Curbside Thai page:

```
<!-- General Information about Curbside Thai -->
```

TIP

Always include comments when working with a team so that you can document the development process for other team members.

A comment can be spread across several lines as long as the comment begins with <!-- and ends with -->. Because comments are ignored by the browser, they can be added anywhere within a document, though it's good practice to always include a comment in the document head in order to describe the document content that follows.

REFERENCE

Adding a Comment to an HTML Document

- To insert a comment anywhere within your HTML document, enter

```
<!-- comment -->
```

where *comment* is the text of the HTML comment.

Add comments to the ct_about.html file indicating the document's author, date of creation, and purpose.

To add a comment to the document:

1. Return to the **ct_about.html** file in your HTML editor.
2. Directly after the opening `<head>` tag, insert the following comment text, indented to make the code easier to read:

HTML comments must be closed with the --> characters.

```
<!--
New Perspectives on HTML 5 and CSS, 8th Edition
Tutorial 1
Tutorial Case
General Information about Curbside Thai
Author: your name
Date: the date

Filename: ct_about.html
-->
```

where ***your name*** is your name and ***the date*** is the current date. Figure 1–10 highlights the newly added comment in the file.

Figure 1–10 Adding a comment to the document

Comment added to the document

```
<head>
   <!--
    New Perspectives on HTML5 and CSS3, 8th Edition
    Tutorial 1
    Tutorial Case
    General Information about Curbside Thai
    Author:   Sajja Adulet
    Date:     3/1/2021
    Filename: ct_about.html
   -->
   <meta charset="utf-8" />
   <meta name="keywords"
         content="Thai, restaurant, Charlotte, food" />
   <title>About Curbside Thai</title>
</head>
```

3. Save your changes to the file.

INSIGHT

Conditional Comments and Internet Explorer

Another type of comment you will encounter in many HTML files is a **conditional comment**, which encloses content that should only be run by particular versions of the Internet Explorer browser. The general form of the conditional comment is

```
<!--[if operator IE version]>
   content
<![endif]-->
```

where *operator* is a logical operator (such as less than or greater than), *version* is the version number of an Internet Explorer browser, and *content* is the HTML code that will be run only if the conditional expression is true. The following code uses the lt (less than) logical operator to warn users that they need to upgrade their browser if they are running Internet Explorer prior to version 8.

```
<!--[if lt IE 8]>
   <p>Upgrade your browser to view this page.</p>
<![endif]-->
```

Other logical operators include `lte` (less than or equal to), `gt` (greater than), `gte` (greater than or equal to), and ! (not). For example, the following code uses the logical operator ! to display the paragraph text only when the browser is *not* Internet Explorer:

```
<!--[if !IE]>
   <p>You are not running Internet Explorer.</p>
<![endif]-->
```

Note that if you omit the version number, the conditional comment is applied to all Internet Explorer versions.

The need for conditional comments arose because Internet Explorer significantly differed from other browsers in how it implemented HTML and there was a need to separate the code meant for the IE browser from code meant for other browsers. This is not as much of a problem with Microsoft ending development of Internet Explorer in favor of the Edge browser, but you may still encounter legacy websites that use conditional comments in their code.

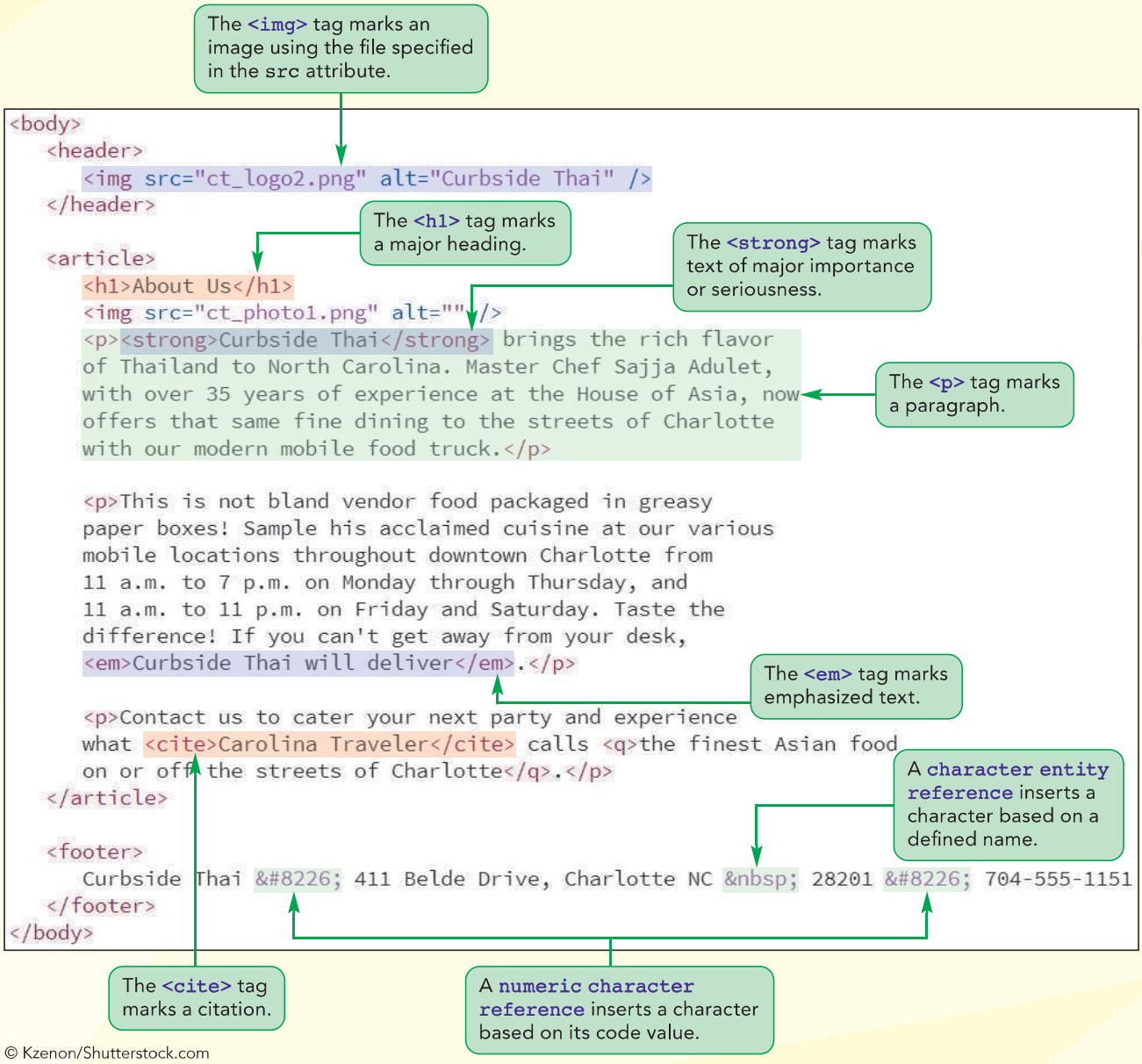
In the next session, you'll continue your work on the ct_about.html file by adding content to the page body.

REVIEW

Session 1.1 Quick Check

1. What is a markup language?
   a. A language used for e-commerce websites
   b. A language describing the content and structure of a document by tagging document elements
   c. A language introduced by Microsoft for use in web browsers
   d. A language that defines the appearance of web pages for computers and cell phones
2. WordPress is a:
   a. web browser
   b. text editor
   c. web content management system
   d. web framework
3. What does W3C stand for?
   a. World Wide Web Computing
   b. World Wide Web Corporation
   c. World Wide Web Creators
   d. World Wide Web Consortium
4. Which of the following is the proper form of an HTML 5 doctype?
   a. `<doctype html>`
   b. `<DOCTYPE html>`
   c. `<doctype>html</doctype>`
   d. `<!DOCTYPE html>`
5. Which of the following defines the page title for an HTML document?
   a. `<title>My Web Page</title>`
   b. `<pageTitle>My Web Page</title>`
   c. `<head id="title">My Web Page</title>`
   d. `<titleStart>My Web Page</titleEnd>`
6. Which of the following is *not* proper HTML syntax?
   a. `<p>Welcome to my <em>web page</em></p>`
   b. `<em><p>Welcome to my web page</em></p>`
   c.
```
<p><em>
   Welcome to my
   web page
</em></p>
```
   d.
```
<p>Welcome to my
<em>web page</em>
</p>
```
7. Which of the following defines "restaurant" as a search keyword for a web page?
   a. `<keyword>restaurant</keyword>`
   b. `<meta>restaurant</meta>`
   c. `<meta name="keywords" content="restaurant" />`
   d. `<meta keyword="restaurant" type="search" />`
8. Which of the following uses the proper syntax for creating an HTML comment?
   a. `<!  Home page for my personal website>`
   b. `<comment>Home page for my personal website</comment>`
   c. `// Home page for my personal website`
   d. `<!-- Home page for my personal website -->`
9. True or false: HTML describes how content should be rendered by the browser.

Session 1.2 Visual Overview:




```
<body>
   <header>
      <img src="ct_logo2.png" alt="Curbside Thai" />
   </header>

   <article>
      <h1>About Us</h1>
      <img src="ct_photo1.png" alt="" />
      <p><strong>Curbside Thai</strong> brings the rich flavor
      of Thailand to North Carolina. Master Chef Sajja Adulet,
      with over 35 years of experience at the House of Asia, now
      offers that same fine dining to the streets of Charlotte
      with our modern mobile food truck.</p>

      <p>This is not bland vendor food packaged in greasy
      paper boxes! Sample his acclaimed cuisine at our various
      mobile locations throughout downtown Charlotte from
      11 a.m. to 7 p.m. on Monday through Thursday, and
      11 a.m. to 11 p.m. on Friday and Saturday. Taste the
      difference! If you can't get away from your desk,
      <em>Curbside Thai will deliver</em>.</p>

      <p>Contact us to cater your next party and experience
      what <cite>Carolina Traveler</cite> calls <q>the finest Asian food
      on or off the streets of Charlotte</q>.</p>
   </article>

   <footer>
      Curbside Thai &#8226; 411 Belde Drive, Charlotte NC   28201 &#8226; 704-555-1151
   </footer>
</body>
```

HTML Page Elements

The opening paragraph of the article is marked with the <p> tag.

Images are added to the web page.

The main heading of the article is marked with the <h1> tag.

The restaurant name is marked with the tag to indicate its importance.

Curbside Thai

About Us

Curbside Thai brings the rich flavor of Thailand to North Carolina. Owner and chef Sajja Adulet, with over 35 years of experience as the award-winning master chef at the House of Asia, now offers that same fine dining to the streets of Charlotte through our modern mobile food truck.

This is not bland vendor food packaged in greasy paper boxes! Sample our acclaimed cuisine at our various mobile locations throughout downtown Charlotte from 11 a.m. to 7 p.m. (M-R) and 11 a.m. to 11 p.m. on Friday and Saturday. Taste the difference! If you can't get away from your desk, *Curbside Thai will deliver.*

Kzenon/Shutterstock.com

Nonbreaking space is inserted with the   character entity reference.

Contact us to cater your next party and experience what *Carolina Traveler* calls "the finest Asian food on or off the streets of Charlotte."

A citation to a magazine is marked with the <cite> tag.

Curbside Thai • 411 Belde Drive, Charlotte NC 28201 • 704-555-1151

An example of emphasized text is marked with the tag.

Bullet characters are inserted with the • numeric character reference.

Writing the Page Body

Now that you have created the document head of the About Curbside Thai web page, you'll begin writing the document body, starting with general markup tags identifying the major sections of the page body, and working inward to more specific content within those sections.

Using Sectioning Elements

The first task in designing the page body is identifying the page's major topics. A page typically has a header, one or more articles that are the chief focus of the page, and a footer that provides contact information for the author or company. HTML marks these major topical areas using the **sectioning elements** described in Figure 1–11.

Figure 1–11 HTML sectioning elements

Element	Description
`address`	Marks contact information for an individual or group
`article`	Marks a self-contained composition in the document such as a newspaper story
`aside`	Marks content that is related to a main article
`body`	Contains the entire content of the document
`footer`	Contains closing content that concludes an article or section
`h1, h2, h3, h4, h5, h6`	Marks major to minor headings with `h1` representing the heading with the highest rank, `h2` representing next highest-ranked heading, and so forth
`header`	Contains opening content that introduces an article or section
`nav`	Marks a list of hypertext or navigation links
`section`	Marks content that shares a common theme or purpose on the page

For example, a news blog page might contain several major topics. To identify these areas, the HTML code for the blog might include the following elements marking off the page's header, navigation list, article, aside, and footer.

```
<body>
   <header>
   </header>
   <nav>
   </nav>
   <article>
   </article>
   <aside>
   </aside>
   <footer>
   </footer>
</body>
```

TIP

Sectioning elements can be nested within each other; an article might contain its own header, footer, and collection of navigation links.

These sectioning elements are also referred to as **semantic elements** because the tag name describes the purpose of the element and the type of content it contains. Even without knowing much about HTML, the page structure defined in the above code is easily understood from the tag names.

REFERENCE

Defining Page Sections

- To mark the page header, use the `header` element.
- To mark self-contained content, use the `article` element.
- To mark a navigation list of hypertext links, use the `nav` element.
- To mark a sidebar, use the `aside` element.
- To mark the page footer, use the `footer` element.
- To group general content, use the `section` element.

The About Curbside Thai page will have a simple structure containing a header, a single article, and a footer. Within the header, there will be an `h1` element providing a main topic heading (not to be confused with the document title, which is displayed on the browser title bar or a browser tab). Add this structure to the document body.

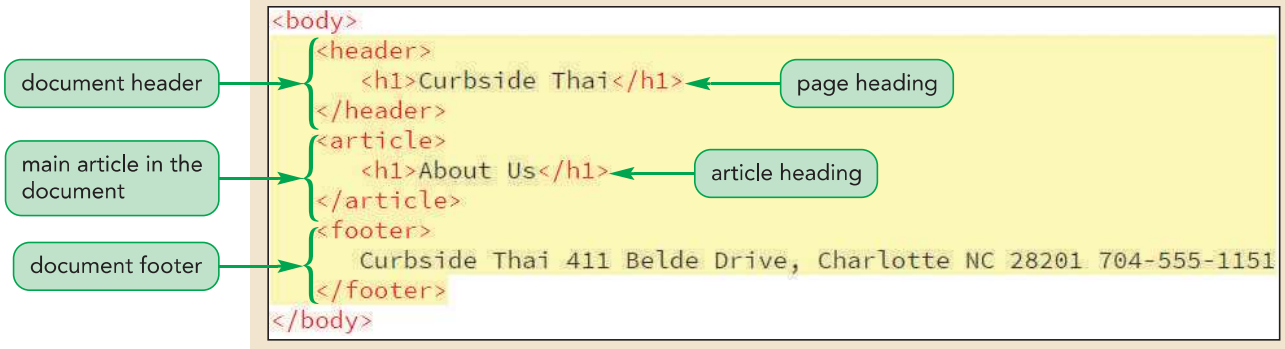
To define the sections in the page body:

1. If you took a break after the previous session, return to the **ct_about.html** file in your HTML editor.
2. Directly after the opening `<body>` tag, insert the following HTML code, indented to make the code easier to read:

```
<header>
   <h1>Curbside Thai</h1>
</header>
<article>
   <h1>About Us</h1>
</article>
<footer>
   Curbside Thai 411 Belde Drive, Charlotte NC 28201 704-555-1151
</footer>
```

Figure 1–12 highlights the sectioning elements used in the page body.

Figure 1–12 Adding sectioning elements to the page body



3. Save your changes to the file.

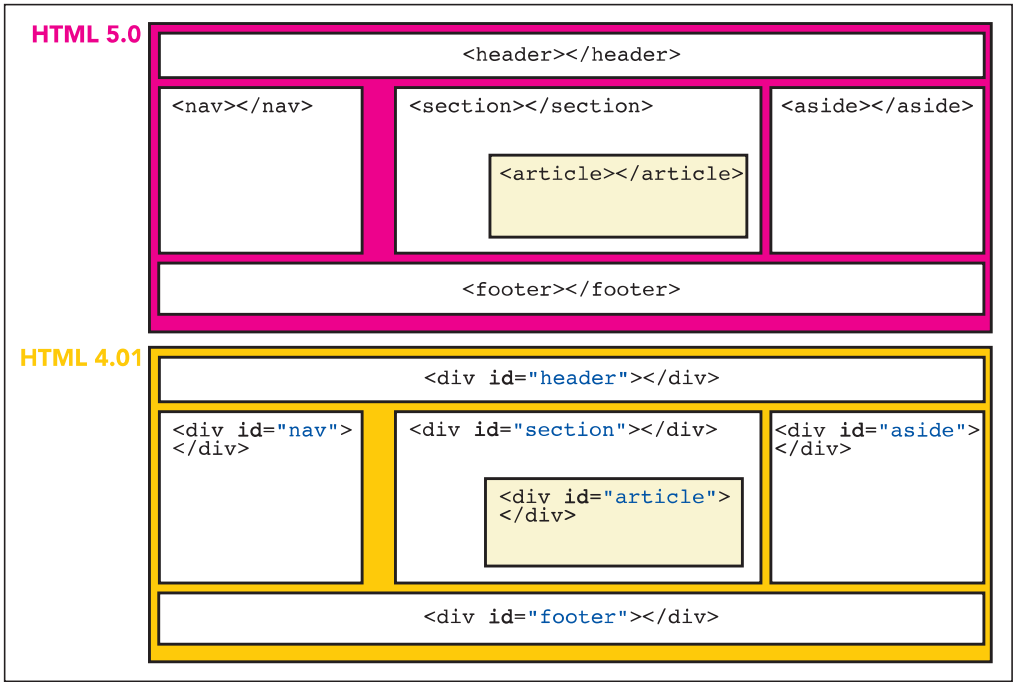
Comparing Sections in HTML 4 and HTML 5

Many of the sectioning elements described in Figure 1–11 were introduced in HTML 5. Prior to HTML 5, sections were defined as divisions created using the following `div` element:

```
<div id="id">
   content
</div>
```

where *id* uniquely identifies the division. Figure 1–13 shows how the same page layout marked up using sectioning elements in HTML 5 would have been defined in HTML 4.01 using `div` elements.

Figure 1–13 Sections in HTML 5.0 vs. divisions in HTML 4.01



One problem with `div` elements is that there are no rules for ids. One web designer might identify the page heading with the id *header,* while another designer might use *heading* or *top*. The lack of consistency makes it harder for search engines to identify the page's main topics. The advantage of the HTML 5 sectioning elements is that their tag name indicates their purpose in the document, leading to greater uniformity in how pages are designed and interpreted. However, you might still encounter use of the `div` element in older websites and within the code generated by web frameworks.

Using Grouping Elements

Within sectioning elements are **grouping elements**. Each grouping element organizes similar content into a distinct group, much like a paragraph groups sentences that share a common theme. Figure 1–14 describes the HTML grouping elements.

Figure 1–14 HTML grouping elements

Element	Description
`blockquote`	Contains content that is quoted from another source, often with a citation and often indented on the page
`div`	Contains a generic grouping of elements within the document
`dl`	Marks a description list containing one or more dt elements with each followed by one or more `dd` elements
`dt`	Contains a single term from a description list
`dd`	Contains the description or definition associated with a term from a description list
`figure`	Contains an illustration, photo, diagram, or similar object that is cross-referenced elsewhere in the document
`figcaption`	Contains the caption associated with a figure
`hr`	Marks a thematic break such as a scene change or a transition to a new topic (often displayed as a horizontal rule)
`main`	Marks the main content of the document or application; only one `main element` should be used in the document
`ol`	Contains an ordered list of items
`ul`	Contains an unordered list of items
`li`	Contains a single item from an ordered or unordered list
`p`	Contains the text of a paragraph
`pre`	Contains a block of preformatted text in which line breaks and extra spaces in the code are retained (often displayed in a monospace font)

The following code shows three paragraphs nested within a page article with each paragraph representing a group of similar content:

```
<article>
   <p>Content of 1st paragraph.</p>
   <p>Content of 2nd paragraph.</p>
   <p>Content of 3rd paragraph.</p>
</article>
```

The default style for browsers is to start sectioning or grouping elements on a new line, separating them from any content that appears before it. Thus, each of these paragraphs will appear in the web page on a new line as will the article itself. Note that the exact appearance of the paragraphs and the space between them depends on the styles applied by the browser to those elements. You'll learn more about styles later in this tutorial.

REFERENCE

Defining Page Groups

- To mark a paragraph, use the `p` element.
- To mark an extended quote, use the `blockquote` element.
- To mark the main content of a page or section, use the `main` element.
- To mark a figure box, use the `figure` element.
- To mark a generic division of page content, use the `div` element.

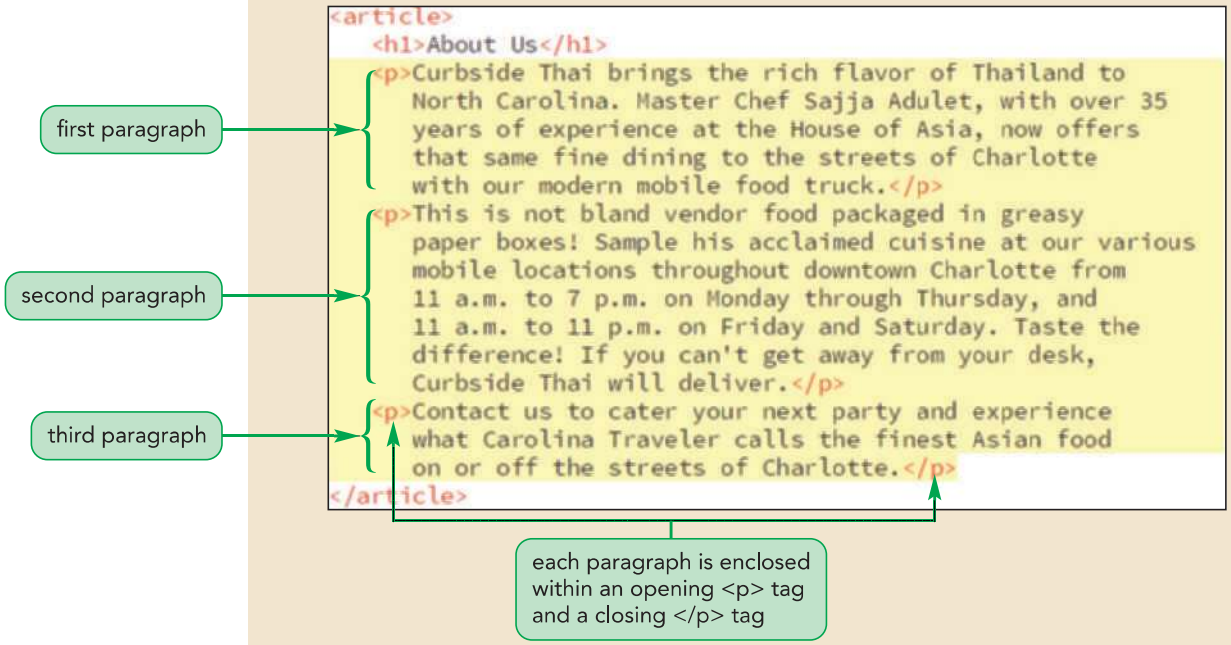
Sajja has written an article describing Curbside Thai. Enter the text of the article into the `article` element in the About Curbside Thai web page and use `p` elements to mark the paragraphs in the article.

To group the page text into paragraphs:

1. Use a text editor to open the **ct_pages.txt** file from the html01 ▸ tutorial folder.
2. Select and copy the three paragraphs of text directly after the About Us title.
3. Close the file, but do not save any changes you may have inadvertently made to the document.
4. Return to the **ct_about.html** file in your HTML editor.
5. Directly after the `<h1>About Us</h1>` line within the page article, insert a new blank line and paste the text you copied.
6. Enclose each of the three paragraphs of pasted content between an opening ** ** tag and a closing ** ** tag. Indent the code within the `article` element to make the code easier to read.

   Figure 1–15 highlights the newly added code for the three paragraphs of article text.

Figure 1–15 Grouping article content by paragraphs



7. Save your changes to the file.

Using Text-Level Elements

Within each grouping element are **text-level elements**, which mark phrases or characters within a paragraph. Unlike sectioning or grouping elements that mark a self-contained block of content, text-level elements appear in line with the surrounding content and are known as **inline elements**. For example, the *italicized* or **boldface** text in this paragraph is considered inline content because it appears alongside the surrounding text. Figure 1–16 describes some of the many text-level elements in HTML.

Figure 1–16 HTML text-level elements

Element	Description
a	Marks content that acts as a hypertext link
abbr	Marks an abbreviation or acronym
b	Indicates a span of text to which attention should be drawn (text usually appears in bold)
br	Represents a line break within the grouping element
cite	Marks a citation to a title or author of a creative work (text usually appears in italics)
code	Marks content that represents computer code (text usually appears in a monospace font)
data	Associates a data value with the marked text with the value attribute providing the value
dfn	Marks a defined term for which a definition is given elsewhere in the document
em	Indicates content that is emphasized or stressed (text usually appears in italics)
i	Indicates a span of text that expresses an alternate voice or mood (text usually appears in italics)
kbd	Marks text that represents user input, typically from a computer keyboard or a voice command
mark	Marks content that is highlighted for reference purposes
q	Marks content that is quoted from another source
s	Marks content that is no longer accurate or relevant (text is usually struck through)
samp	Marks text that represents the sample output from a computer program or application
small	Marks side comments (text usually in small print)
span	Contains a generic run of text within the document
strong	Indicates content of strong importance or seriousness (text usually appears in bold)
sub	Marks text that should be treated as a text subscript
sup	Marks text that should be treated as a text superscript
time	Marks a time value or text string
u	Indicates text that appears stylistically different from normal text (text usually appears underlined)
var	Marks text that is treated as a variable in a mathematical expression or computer program
wbr	Represents where a line break should occur, if needed, for a long text string

The following HTML code demonstrates how to employ text-level elements to mark select phrases or characters within a paragraph.

```
<p>
   Contact us to cater your next party and experience what
   <cite>Carolina Traveler</cite> calls <q>the finest
   Asian food on or off the streets of Charlotte.</q>
</p>
```

Two text-level elements are used in this paragraph: the `cite` element marks the citation to the *Carolina Traveler* magazine and the `q` element marks the direct quote from the magazine's review of Curbside Thai. Both the citation and the quoted material will appear alongside other text within the paragraph.

REFERENCE

Defining Text-Level Content

- To mark emphasized text, use the `em` element.
- To mark text of great importance, use the `strong` element.
- To mark a citation, use the `cite` element.
- To mark a selection of quoted material, use the `q` element.
- To mark a subscript, use the sub element; to mark a superscript, use the `sup` element.
- To mark a generic selection of text-level content, use the `span` element.

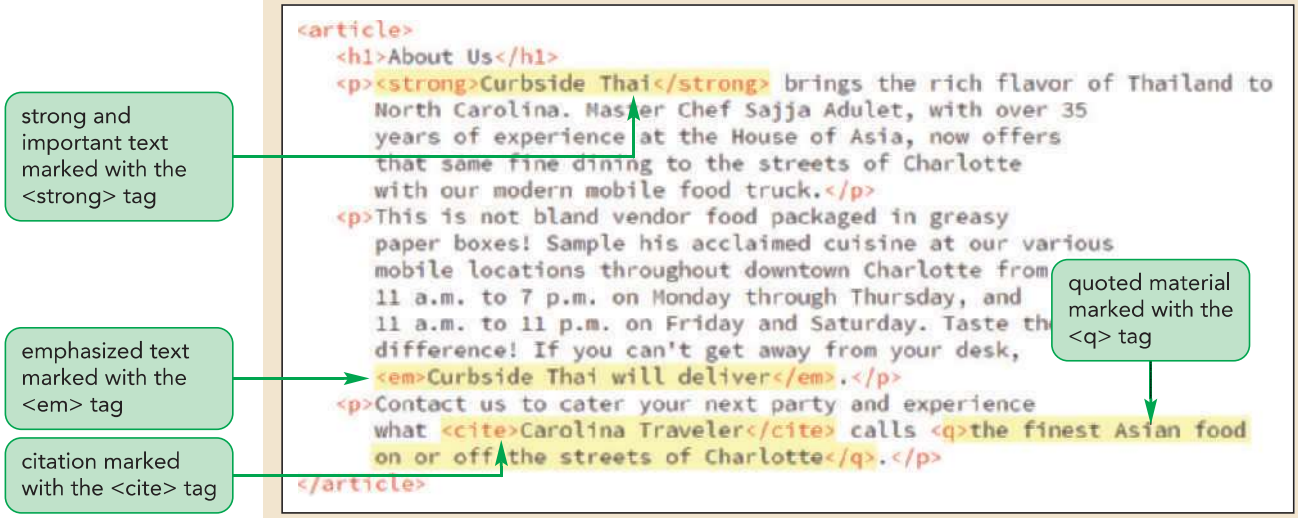
Use text-level elements in the About Curbside Thai web page to mark examples of emphasized text, strongly important text, citations, and quoted material.

To apply text-level elements to a page:

1. Go to the first paragraph within the page article and enclose the opening words *Curbside Thai* within a set of opening and closing **** tags. Use `<strong>` tags when you want to strongly reinforce the importance of the text, such as the restaurant name, for the reader.
2. In the second paragraph, enclose the phrase, *Curbside Thai will deliver* within a set of opening and closing **** tags to emphasize this text.
3. Go the third paragraph and mark *Carolina Traveler* using the cite element and then mark the extended quote, *the finest Asian food on or off the streets of Charlotte*, using the `q` element.

   Figure 1–17 highlights the application of the four text-level elements to the paragraph text.

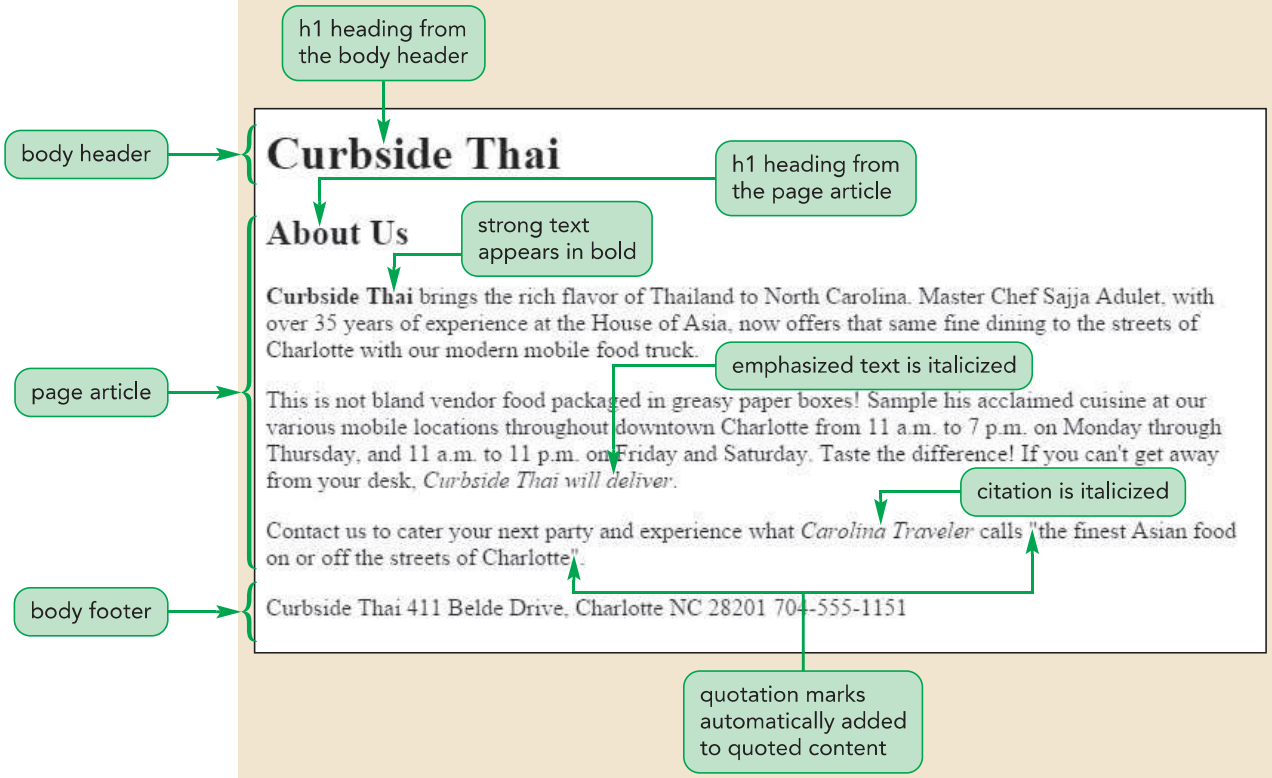
Figure 1–17 **Marking text-level content**



4. Save your changes to the file.
5. Open the **ct_about.html** file in your browser to view how your browser renders the page content.

   Figure 1–18 shows the current appearance of the page.

Figure 1–18 The About Curbside Thai page as rendered by the browser



Trouble? Depending on your browser, device, and screen settings, you might see some minor differences in the appearance of the About Curbside Thai web page from that shown in Figure 1–18.

In rendering the page, the browser made the following stylistic choices for the different page elements:

- The h1 heading from the body header is assigned the largest font and is displayed in bold to emphasize its importance. The h1 heading from the page article is given a slightly smaller font but is still displayed in bold.
- Strong text is displayed in bold while emphasized text is displayed in italics.
- Citations are displayed in italic while quoted material is automatically surrounded by quotation marks.

TRY IT

You can explore the impact of different HTML tags using the demo_html.html file in the html01 ▸ demo folder.

It needs to be emphasized again that all of these stylistic choices are not determined by the markup tags; they are default styles used by the browser. Different browsers and different devices might render these page elements differently. To exert more control over your page's appearance, you can apply a style sheet to the document.

Linking an HTML Document to a Style Sheet

A **style sheet** is a set of rules specifying how page elements are displayed. Style sheets are written in the **Cascading Style Sheets** (**CSS**) language. Like HTML, the CSS language was developed and enhanced as the web grew and changed and, like HTML, CSS specifications are managed by the W3C.To replace the browser's internal style sheet with one of your own, you can link your HTML file to a style sheet file using the following `link` element:

```
<link href="file" rel="stylesheet" />
```

where *file* is a text file containing the CSS style sheet. Because the `link` element can also be used to link to data other than style sheets, the rel attribute is required to tell the browser that it is linking to style sheet data. Older websites might include `type="text/css"` as part of the `link href` element.

REFERENCE

Linking an HTML Document to an External Style Sheet

- To link an HTML document to an external style sheet file, add the following element to the document head:

```
<link href="file" rel="stylesheet" />
```

where *file* is a text `file` containing the CSS style rules.

TIP

Because the link element is metadata, it's always added to the document head.

Sajja has supplied you with two CSS files that he wants applied to his website. The ct_base.css file contains styles specifying the appearance of text-level elements. The ct_layout2.css file contains styles that govern the arrangement of sectioning and grouping elements on the page. Link the ct_about.html file to both of these style sheets now.

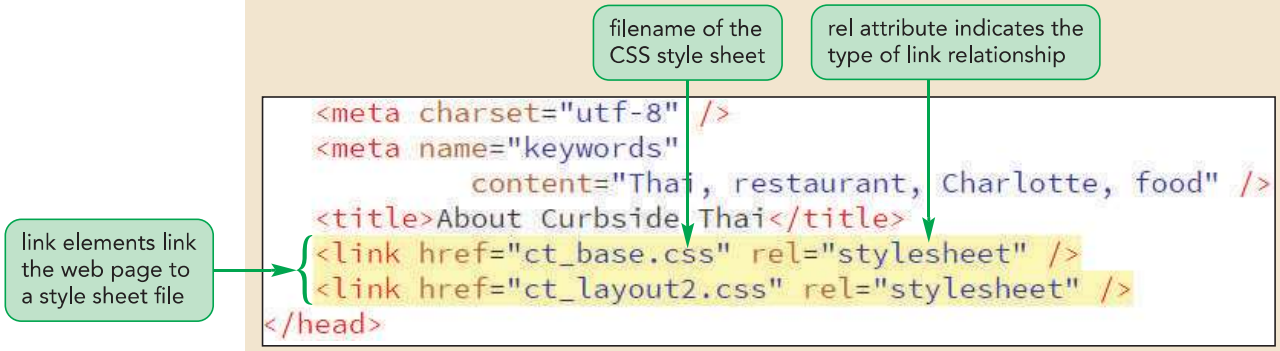
To link an HTML document to a style sheet:

1. Return to the **ct_about.html** file in your HTML editor.
2. Directly before the closing `</head>` tag, insert the following `link` elements:

```
<link href="ct_base.css" rel="stylesheet" />
<link href="ct_layout2.css" rel="stylesheet" />
```

Figure 1–19 highlights the two style sheet links added to the document.

Figure 1–19 Linking to style sheets



3. Save your changes to the file and then reload the ct_about.html file in your browser. Figure 1–20 shows the new appearance of the page using the style sheets provided by Sajja.

Figure 1–20 The About Curbside Thai page rendered under a new style sheet

Applying these style sheets displays the page body on a tan background with a drop shadow, the font used in the two h1 headings has changed, a top border has been added to the footer to set it off from the preceding content, and the citation to the *Carolina Traveler* magazine is displayed in a light blue font. The effect makes the page content easier to read and more pleasing to the eye.

Sajja is concerned that the contact information in the page footer is difficult to read. He wants you to add bullet characters (•) separating the name of the restaurant, the street address, and the restaurant phone number. However, this character is not represented by any keys on your keyboard. How then, do you insert this symbol into the web page?

Working with Character Sets and Special Characters

Every character that your browser is capable of rendering belongs to a collection of characters and symbols called a **character set**. The character set used for the English alphabet is the **American Standard Code for Information Interchange** more simply known as **ASCII**. A more extended character set, called **Latin-1** or the **ISO 8859-1** character set, supports 255 characters and can be used by most languages that employ the Latin alphabet, including English, French, Spanish, and Italian. **Unicode**, the most extended character set, supports up to 65,536 symbols and can be used with any of the world's languages.

Character Encoding

TRY IT

You can explore different character encoding values using the demo_characters.html file in the html01 ▸ demo folder.

Each character from a character set is associated with an encoding value that can then be stored and read by a computer program. For example, the copyright symbol © from the Unicode character set is encoded with the number 169. If you know the encoding value, you can insert the corresponding character directly into your web page using the following character encoding reference:

```
&#code;
```

where *code* is the encoding reference number. Thus, to display the © symbol in your web page, you would enter

```
&#169;
```

into your HTML file.

Character Entity References

Another way to insert a special symbol is to insert a character entity reference, which is a short memorable name used in place of the encoding reference number. Character entity references are inserted as

```
&char;
```

where *char* is the character's entity reference. The character entity reference for the copyright symbol is `copy`, so to display the © symbol in your web page, you could insert the following expression into your HTML code:

```
&copy;
```

In the last session, you learned that HTML will collapse consecutive occurrences of white space into a single white-space character. You can force HTML to display extra white space by using the following character entity reference

```
 
```

where `nbsp` stands for *nonbreaking space*. When you want to display extra white space, you need to insert the nonbreaking space character reference in the HTML code for each space you want to display.

REFERENCE

Inserting Symbols from a Character Set

- To insert a symbol based on the character encoding reference number, enter

  ```
  &#code;
  ```

  where *code* is the character encoding reference number.
- To insert a symbol based on a character entity reference, enter

  ```
  &char;
  ```

  where *char* is the name assigned to the character.
- To insert a white-space character, use

  ```
   
  ```

For the footer in the About Curbside Thai page, use the bullet symbol (•), which has the encoding value 8226, to separate the restaurant name, address, and phone number. Use the ` ` character reference to insert an extra blank space prior to the postal code in the restaurant address.

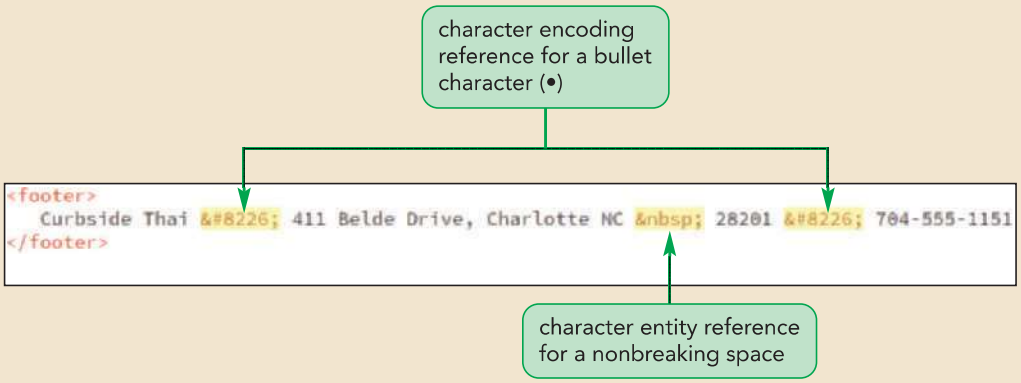
To insert a character encoding reference number and an entity reference:

1. Return to the **ct_about.html** file in your HTML editor.
2. Go to the `footer element` and insert the character encoding number **`&#8226;`** directly after the word *Thai* and after the postal code *28201*. Insert the character reference **` `** directly before the postal code.

   Figure 1–21 highlights the character codes and references added to the footer.

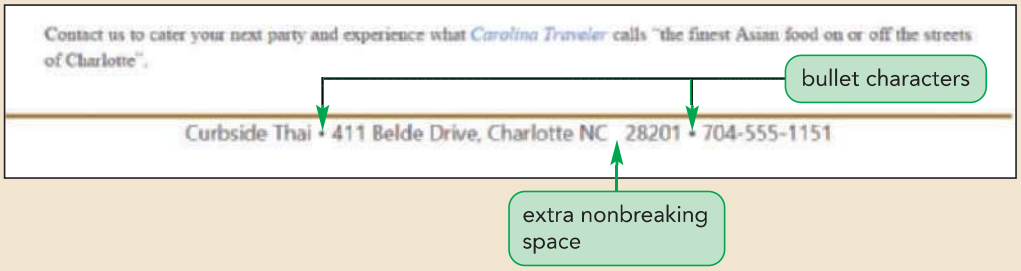
Character encoding reference numbers must always begin with &# and end with a semicolon, otherwise the code won't be recognized as a code number.

Figure 1–21 Inserting special characters



3. Save your changes to the file and then reload the ct_about.html file in your browser. Confirm that the footer now shows the characters displayed in Figure 1–22.

Figure 1–22 Revised page footer



INSIGHT

Presentational Attributes

Early versions of HTML supported **presentational elements** and **presentational attributes** describing how each element should be rendered by the browser. For example, to align text on a page, HTML included the following `align` attribute

```
<element align="alignment">content</element>
```

where *alignment* is either `left`, `right`, `center`, or justify. Thus, to center an h1 heading on a page, apply the following code:

```
<h1 align="center">Curbside Thai</h1>
```

Almost all presentational elements and attributes are now deprecated in favor of style sheets, but you may still see them in the code from older websites. Using a deprecated attribute like `align` would probably not cause your web page to fail, however, it's still best practice to adhere to a standard in which HTML is used only to describe the content and structure of the document and style sheets are used to format its appearance.

So far your work on the Curbside Thai page has been limited to textual content. Next, you'll explore how to add graphical content to your web page.

Working with Inline Images

Most web pages include **embedded content**, which is content imported from another resource, often nontextual, such as graphic images, audio soundtracks, video clips, or interactive games. To support this type of content, HTML provides the **embedded elements** listed in Figure 1–23.

Figure 1–23 HTML embedded elements

Element	Description
`audio`	Represents a sound clip or audio stream
`canvas`	Contains programming scripts used to construct bitmap images and graphics
`embed`	Contains general embedded content including application or interactive content
`iframe`	Contains the contents of an external web page or Internet resource
`img`	Contains a graphic image retrieved from an image file
`object`	Contains general embedded content including application or interactive content
`video`	Represents a video clip or video stream with captions

TIP

Always include the `alt` attribute; it is required in XHTML code and is highly recommended as a way of accommodating users running nonvisual web browsers.

These elements are also known as **interactive elements** because they allow for interaction between the user and the embedded object. For example, embedded audio or video content usually contains player buttons to control the playback.

Images are inserted into a web page using the following `img` element

```
<img src="file" alt="text" />
```

where *file* is the name of the image file. If the browser cannot display images, the text in the `alt` attribute is used in place of the image. As with other one-sided tags, the `img` element can be entered without the closing slash as

```
<img src="file" alt="text">
```

Images marked with the `<img>` tag are also known as **inline images** because they are placed, like text-level elements, in line with surrounding content.

By default, the image size matches the size of the image in the file but you can specify a different size by adding the following `width` and `height` attributes to the `img` element

```
width="value" height="value"
```

where the `width` and `height` values are expressed in pixels. If you specify only the width, browsers automatically set the height to maintain the proportions of the image; similarly, if you define the height, browsers automatically set the width to maintain the image proportions. Image sizes can also be set within the document's style sheet.

REFERENCE

Embedding an Inline Image

- To embed an inline image into the document, use

```
<img src="file" alt="text" />
```

where *file* is the name of the graphic image file, and *text* is text displayed by browsers in place of the graphic image.

Sajja has provided you with two images. The image from the ct_logo2.png file displays the restaurant logo, while the ct_photo1.png image provides an image of customers being served by an employee at his brick-and-mortar restaurant. Sajja included this image to emphasize that the food from his food truck is the same quality and great taste as the food at his award winning restaurant. Add both of these images to the ct_about.html file.

To insert inline images into a document:

1. Return to the **ct_about.html** file in your HTML editor.
2. Go to the header element and replace the h1 element with the tag

```
<img src="ct_logo2.png" alt="Curbside Thai" />
```

TIP

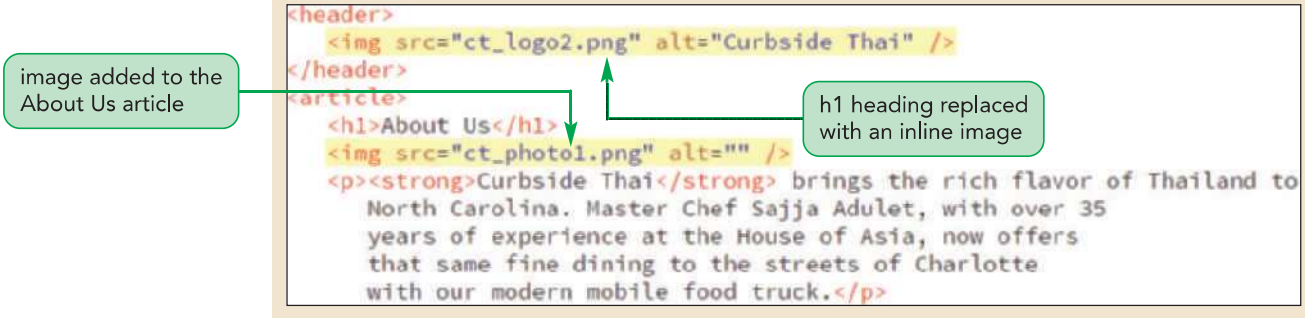
Include the alt attribute as a blank text string if the image file does not convey any text message to the user.

3. Go to the article element and, directly after the `h1` element, insert the tag

```
<img src="ct_photo1.png" alt="" />
```

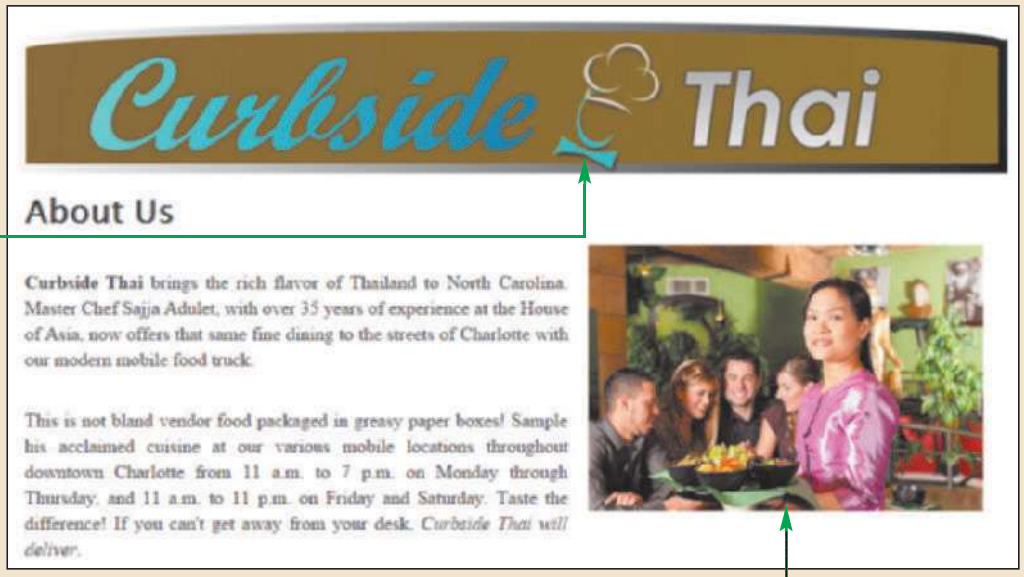
Figure 1–24 highlights the newly added `img` elements in the document.

Figure 1–24 Inserting inline images



4. Save your changes to the file and then reload the ct_about.html file in your browser. Figure 1–25 displays the newly added graphic images in the web page.

Figure 1–25 **Images on the About Curbside Thai page**



restaurant logo used for the page header

photo floated on the right margin of the article

Trouble? The exact appearance of the text as it flows around the image will vary depending on the width of your browser window.

Note that the photo of the Curbside Thai customers is floated alongside the right margin of the article, with the surrounding paragraphs flowing around the image. This is the result of code in the style sheets. You'll learn about styles used to float images in Tutorial 3.

Line Breaks and Other Empty Elements

The `img` element is inserted using the empty element tag because it does not enclose any text content, but instead links to an external image file. Another important empty element is the following br element, which creates a line break

```
<br />
```

Line breaks are placed within grouping elements, such a paragraphs or headings, to force page content to start on a new line within the group. While useful for controlling the flow of text within a group, the `br` element should not be used as a formatting tool. For example, it would not make semantic sense to insert two or more br elements in a row if the only reason to do so is to increase the spacing between lines of text. Instead, all such formatting choices belong in a style sheet.

If the text of a line cannot fit within the width of the viewing window, the browser will wrap the text automatically at the point the browser identifies as the most appropriate. To recommend a different line break point, use the `wbr` (word break) element to indicate where a line break should occur if needed. For example, the following HTML code uses the `wbr` element to break a long web address between ".com/" and "general", but this break happens only if the address will not fit on one line.

```
www.curbsidethai.com/<wbr />general/docs/ct_about.html
```

Finally, another oft-used empty element is the following `hr` or horizontal rule element, which denotes a major topic change within a section

```
<hr />
```

Originally, the `hr` element was used to insert horizontal lines into the page and, although that task is better left to style sheets, you will still see the `hr` element used in that capacity in older web pages.

INSIGHT

Supporting HTML 5 with Legacy Browsers

HTML 5 introduced several new semantic elements including the `header`, `footer`, `article`, and `nav` elements. Some browsers, such as Internet Explorer Version 8, could not cope with new elements without an external program known as a **script** running in the browser.

One script that provides support for HTML 5 is **Modernizr** (*http://modernizr.com*); another is **HTML 5 Shiv** (*https://github.com/aFarkas/html5shiv*). Many HTML editors, such as Dreamweaver, supply their own script files to cope with legacy browsers. Note that even with these scripts, the rendering of your page under old browsers might not match current browsers.

Working with Block Quotes and Other Elements

Now that you've written the code for the ct_about.html file, you'll work on other pages in the Curbside Thai website. The ct_reviews.html file provides excerpts of reviews from food critics and magazines. Because these excerpts contain extended quotes, you'll place each review in the following `blockquote` element

```
<blockquote>
   content
</blockquote>
```

where *content* is the text of the quote. By default, most browsers render block quotes by indenting the quoted material to separate from it from the website author's words, however, you can substitute your own style with a custom style sheet.

Sajja has created much of the code required for the reviews page. The code is contained in the two style sheets that are already linked to the reviews page. Complete the page by adding the excerpts of the reviews marked as block quotes.

To create the reviews page:

1. Open the **ct_reviews_txt.html** file from the html01 ▸ tutorial folder in your HTML editor. Enter ***your name*** and ***the date*** in the comment section and save the file as **ct_reviews.html**.
2. Go to the **ct_pages.txt** file in your text editor.
3. Locate the section containing the restaurant reviews and copy the text of the two reviews and two awards.
4. Return to the **ct_reviews.html** file in your HTML editor and paste the text of the four reviews directly after the `<h1>Reviews</h1>` line.

5. Enclose each review within a set of `<blockquote>` tags. Enclose each paragraph within each review with a set of `<p>` tags. Align and indent your code to make it easier to read.

   Figure 1–26 highlights the newly added code in the document.

Figure 1–26 Marking extended text as block quotes

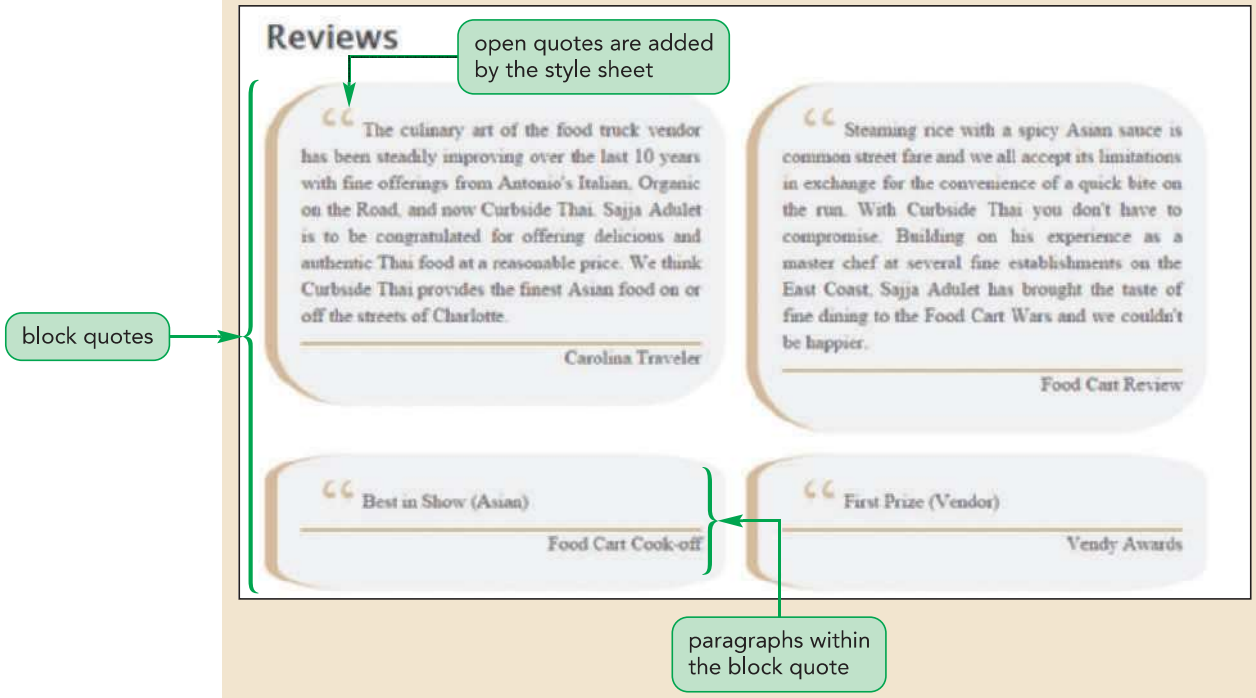
```
<article>
   <h1>Reviews</h1>
   <blockquote>
      <p>The culinary art of the food truck vendor has been steadily
      improving over the last 10 years with fine offerings from
      Antonio's Italian, Organic on the Road, and now Curbside Thai.
      Sajja Adulet is to be congratulated for offering delicious
      and authentic Thai food at a reasonable price. We think
      Curbside Thai provides the finest Asian food on or off
      the streets of Charlotte.</p>
      <p>Carolina Traveler</p>
   </blockquote>
   <blockquote>
      <p>Steaming rice with a spicy Asian sauce is common street fare
      and we all accept its limitations in exchange for the
      convenience of a quick bite on the run. With Curbside Thai you
      don't have to compromise. Building on his experience as a
      master chef at several fine establishments on the East Coast,
      Sajja Adulet has brought the taste of fine dining to the
      Food Cart Wars and we couldn't be happier.</p>
      <p>Food Cart Review </p>
   </blockquote>
   <blockquote>
      <p>Best in Show (Asian)</p>
      <p>Food Cart Cook-off</p>
   </blockquote>
   <blockquote>
      <p>First Prize (Vendor)</p>
      <p>Vendy Awards</p>
   </blockquote>
</article>
```

blockquote elements within the page body

within each block quote are paragraphs

6. Save your changes to the file and then open the **ct_reviews.html** file in your browser. Figure 1–27 shows the appearance of the restaurant review quotes using Sajja's style sheet.

Figure 1–27 Block quotes of restaurant reviews



Because of the styles in Sajja's style sheets, each `blockquote` element appears within its own formatted box with an opening quote character added to reinforce the fact that this is quoted material.

The next page you'll create contains information about catering from Curbside Thai. The structure of this page is identical to the structure of the About Curbside Thai page. Sajja has linked the catering page to two style sheets containing the style rules that dictate how the page will look when the page is rendered in a browser.

To create the Catering page:

1. Open the **ct_catering_txt.html** file from the html01 ▸ tutorial folder in your HTML editor. Enter ***your name*** and ***the date*** in the comment section and save the file as **ct_catering.html**.
2. Return to the **ct_pages.txt** file in your text editor.
3. Locate the section containing information about Curbside Thai's catering service and copy the four paragraphs of information.
4. Return to the **ct_catering.html** file in your HTML editor and paste the copied text directly after the `<h1>Catering</h1>` line.
5. Mark each paragraph in the article using the p element. Align and indent your code to make it easier to read.
6. Directly after the `<h1>Catering</h1>` tag, insert an inline image using **ct_photo2.png** as the source and an empty text string for the `alt` attribute.

   Figure 1–28 highlights the newly added paragraphs in the document.

Figure 1–28 Entering the markup for the Catering page

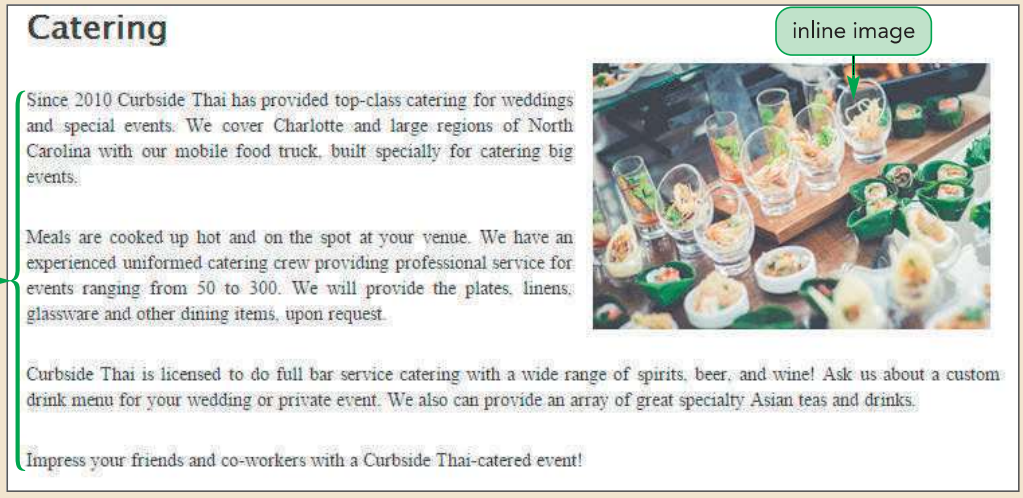
inline image

paragraphs

```
<article>
   <h1>Catering</h1>
   <img src="ct_photo2.png" alt="" />
   <p>Since 2010 Curbside Thai has provided top-class catering for weddings
   and special events. We cover Charlotte and large regions of North
   Carolina with our mobile food truck, built specially for catering big
   events.</p>
   <p>Meals are cooked up hot and on the spot at your venue. We have an
   experienced uniformed catering crew providing professional service for
   events ranging from 50 to 300. We will provide the plates, linens,
   glassware and other dining items, upon request.</p>
   <p>Curbside Thai is licensed to do full bar service catering with a wide
   range of spirits, beer, and wine! Ask us about a custom drink menu
   for your wedding or private event. We also can provide an array of
   great specialty Asian teas and drinks.</p>
   <p>Impress your friends and co-workers with a Curbside Thai-catered
   event!</p>
</article>
```

7. Save your changes to the file and then open the **ct_catering.html** file in your browser. Figure 1–29 shows the appearance of the page.

Figure 1–29 Content of the Catering page



RojaninSri/Shutterstock.com

The final page you'll create in this session will contain contact information for Curbside Thai. Mark the content within the main page article.

To create the Contact Us page:

1. Open the **ct_contact_txt.html** file from the html01 ▸ tutorial folder in your HTML editor. Enter ***your name*** and ***the date*** in the comment section and save the file as **ct_contact.html**. Note that this page is linked to two style sheets that Sajja created.
2. Go to the **ct_pages.txt** file in your text editor.
3. Copy the Contact Us section in the text file (excluding the title).

4. Return to the **ct_contact.html** file in your HTML editor and paste the copied text directly after the `<h1>Contact Us</h1>` tag.
5. Enclose the introductory paragraph within a set of opening and closing **<p>** tags to mark it as a paragraph.
6. Enclose the three lines containing the street address within a set of opening and closing `<address>` tags to mark that content as an address. Insert the **
** tag at the end of the first two lines to create a line break between the name of the restaurant and the street address.
7. Mark each of the last two lines as paragraphs using the p element.

   Figure 1–30 highlights the marked up code for Curbside Thai's contact information.

Figure 1–30 Entering the markup for the Contact Us page

address element to mark up a mailing address

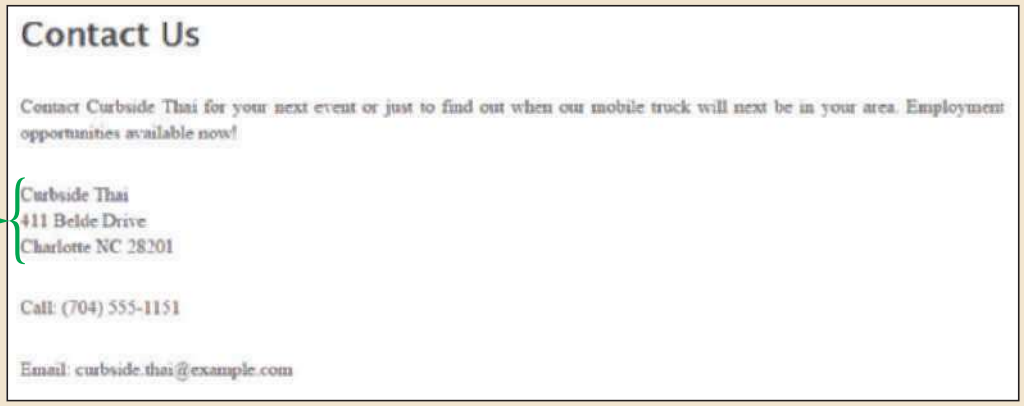
line breaks to start the next part of the address on a new line

```
<article>
   <h1>Contact Us</h1>
   <p>Contact Curbside Thai for your next event or just to find
   out when our mobile truck will next be in your area.
   Employment opportunities available now!</p>

   <address>
      Curbside Thai<br />
      411 Belde Drive<br />
      Charlotte NC 28201
   </address>

   <p>Call: (704) 555-1151</p>
   <p>Email: curbside.thai@example.com </p>
</article>
```

8. Save your changes to the file and then open the **ct_contact.html** file in your browser as shown in Figure 1–31.

Figure 1–31 Content of the Contact Us page

street address



The Contact Us page only provides the text of the contact information but that text is static. In the next session, you'll learn how to make this content interactive by turning the contact information into hypertext.

PROSKILLS

Problem Solving: Making your Page Accessible with ARIA

The web is for everyone and that presents a special challenge when writing code for the visually impaired who will be accessing your website with a screen reader. One standard to assist screen readers is **Accessible Rich Internet Applications** (**ARIA**), which supplements HTML elements with additional attributes that provide clues as to the element's purpose as well as provide information on the current status of every page element.

One of the cornerstones of ARIA is the role attribute, which specifies the purpose of a given element. For example, the following `role` attribute indicates that the `header` element contains a banner, such as a logo that introduces the web page

```
<header role="banner">
   content
</header>
```

ARIA supports a list of approved role names including the following:

- alert — Content with important and usually time-sensitive information
- application — A web application, as opposed to a web document
- definition — A definition term or concept
- dialog — An application window that will require user input
- log — A region of data that is constantly modified and updated
- progress bar — Content that displays the progress status for ongoing tasks
- search — Content that provides search capability to the user
- separator — A divider that separates one region of content from another
- timer — A region that contains a numerical counter reporting on elapsed time

You can view the complete list of role attribute values and how to apply them at *www.w3.org/WAI/PF/aria/roles*.

ARIA is a useful tool for enhancing the accessibility of your web page and making the rich resource that is the World Wide Web open to all. A side benefit is that accessibility and usability go hand-in-hand. A website that is highly accessible is also highly usable and that is of value to all users.

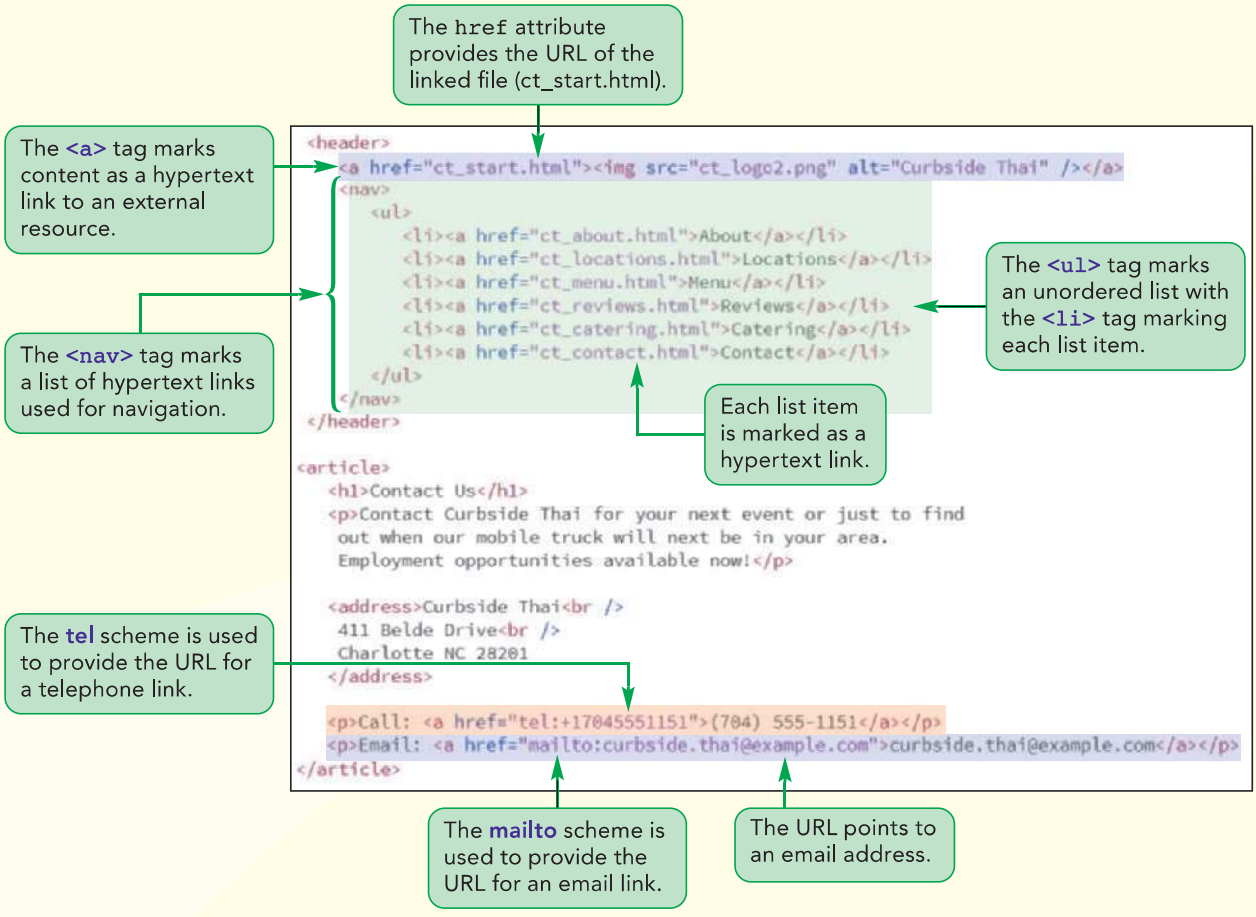
In the next session, you'll continue to work on the Curbside Thai website by adding pages describing the restaurant menu and listing the time and locations where the mobile food truck is parked.

REVIEW

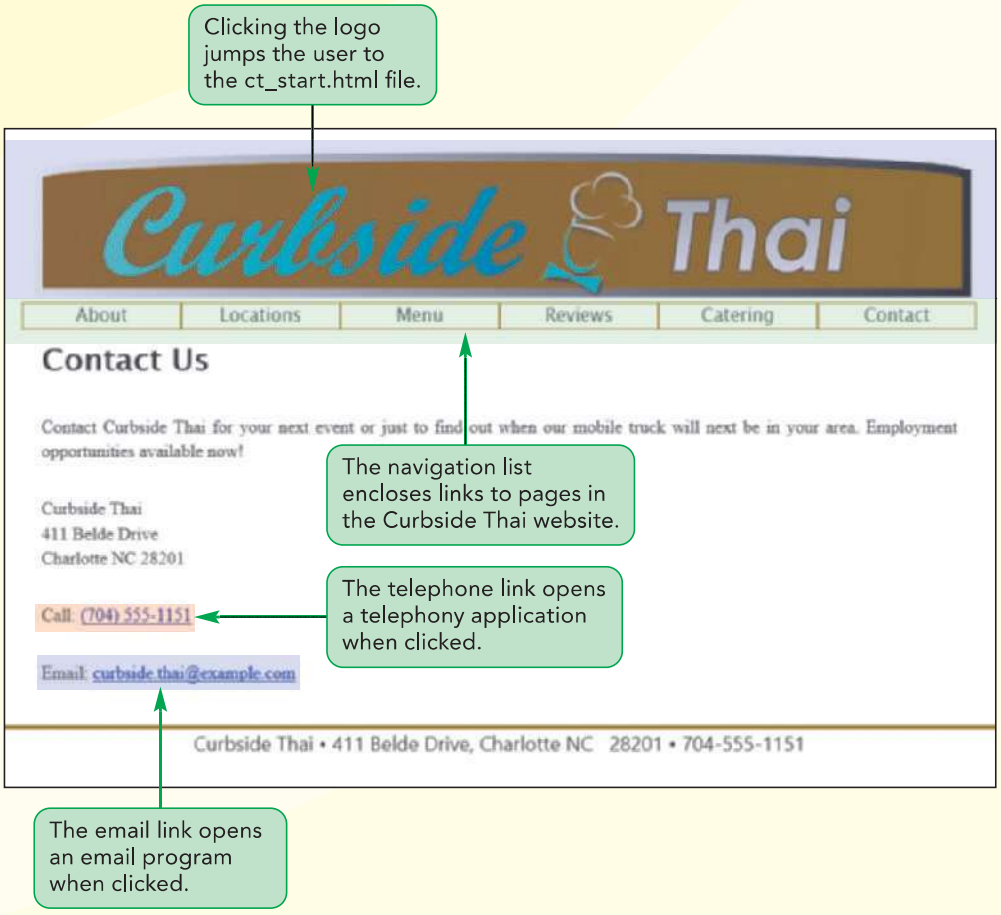
Session 1.2 Quick Check

1. Which of the following marks a heading with the second level of importance?
   a. `<h level="2">Gourmet Thai Cooking</h>`
   b. `<hLevel2>Gourmet Thai Cooking</hLevel2>`
   c. `<heading2>Gourmet Thai Cooking</heading2>`
   d. `<h2>Gourmet Thai Cooking</h2>`
2. Prior to HTML 5, sections were identified with:
   a. the `section` element and the `name` attribute
   b. the `div` element and the `id` attribute
   c. the `article` element and the `section` attribute
   d. the `aside` element and the `id` attribute
3. To mark content that is related to the main article, use
   a. the `sidebar` element
   b. the `extraContent` element
   c. the `aside` element
   d. the `section` element
4. Which of the following marks the content as emphasized text?
   a. `<i>Daily Special</i>`
   b. `<em>Daily Special</em>`
   c. `<strong>Daily Special</strong>`
   d. `<b>Daily Special</b>`
5. Which of the following should be used to mark the text string "H_2SO_4"?
   a. `H<lower>2</lower>SO<lower>4</lower>`
   b. `H<sup>2</sup>SO<sup>4</sup>`
   c. `H<code>2</code>SO<code>4</code>`
   d. `H<sub>2</sub>SO<sub>4</sub>`
6. Which of the following links the HTML file to the CSS style sheet file, mystyles.css?
   a. `<stylesheet rel=link">mystyles.css</mystyles>`
   b. `<stylesheet src="mystyles.css" />`
   c. `<link rel="stylesheet">mystyles.css</link>`
   d. `<link href="mystyles.css" rel="stylesheet" />`
7. Which of the following tags might be used to indicate a change of topic within a section?
   a. `<hr />`
   b. `<br />`
   c. `<aside />`
   d. `<img />`
8. Which character reference is used to insert a nonbreaking space within a text string?
   a. `&space;`
   b. ` `
   c. `&ws;`
   d. `&copy;`
9. Which of the following tags inserts an inline image with the alternative text "Art World"?
   a. `<image src="awlogo.png" alt="Art World" />`
   b. `<img src="awlogo.png">Art World</img>`
   c. `<figure src="awlogo.png" alt="Art World" />`
   d. `<img src="awlogo.png" alt="Art World" />`

Session 1.3 Visual Overview:



Lists and Hypertext Links



Working with Lists

In the last session, you added order and structure to your web page with some of HTML's sectioning and grouping elements. Another type of grouping element is a list. HTML supports three types of lists: ordered lists, unordered lists, and description lists.

Ordered Lists

Ordered lists are used for items that follow some defined sequential order, such as items arranged alphabetically or numerically. An ordered list is marked using the `ol` (ordered list) element with each list item marked using the `li` element. The general structure is

```
<ol>
   <li>item1</li>
   <li>item2</li>
   ...
</ol>
```

where *item1*, *item2*, and so forth are the items in the list. For example, the following ordered list ranks the top-three most populated states:

```
<ol>
   <li>California</li>
   <li>Texas</li>
   <li>New York</li>
</ol>
```

By default, browsers display list items alongside a numeric marker. In the case of ordered lists, this is a numeric value starting with the number 1 and ascending in value. For example, the ordered list of states would be rendered in most browsers as

1. California
2. Texas
3. New York

Note that because both the `ol` and `li` elements are considered grouping elements, each list item will appear, by default, on a new line in the document.

To display different numbering, use the `start` and `reversed` attributes of the `ol` element. The `start` attribute provides the numeric value for the first item in the list, while the `reversed` attribute specifies that the list numbers should be displayed in descending order. Thus, the following HTML code that lists the most populated states

```
<ol reversed start="50">
   <li>California</li>
   <li>Texas</li>
   <li>New York</li>
</ol>
```

would be rendered as a list in descending order starting from 50

50. California
49. Texas
48. New York

You can explicitly define the item value by adding the `value` attribute to each list item. The list shown previously could also have been generated with the following code:

```
<ol>
   <li value="50">California</li>
   <li value="49">Texas</li>
   <li value="48">New York</li>
</ol>
```

You can use style sheets to display lists using alphabetical markers (A, B, C, …) or Roman Numerals (I, II, III, …) in place of numeric values. You'll explore this technique in Tutorial 2.

Unordered Lists

Unordered lists are lists in which the items have no sequential order. The structure for an unordered list is similar to that used with ordered lists except that the list items are grouped within the following `ul` (unordered list) element:

```
<ul>
   <li>item1</li>
   <li>item2</li>
   …
</ul>
```

For example, the following HTML code creates an ordered list of all of the states along the Pacific coast:

```
<ul>
   <li>California</li>
   <li>Oregon</li>
   <li>Washington</li>
</ul>
```

By default, browsers display items from an unordered list alongside a marker such as a bullet point. Thus, an unordered list of Pacific coast states might be rendered as

- California
- Oregon
- Washington

Once again, the exact appearance of an unordered list will depend on the style sheet that is applied to the element.

INSIGHT

Creating a Nested List

Because the `li` element is itself a grouping element, it can be used to create a series of **nested lists**. The general structure for a nested collection of unordered list is

```
<ul>
   <li>Item 1</li>
   <li>Item 2
      <ul>
         <li>Sub Item 1</li>
         <li>Sub Item 2</li>
      </ul>
   </li>
</ul>
```

where *Sub Item 1*, *Sub Item 2*, and so forth are items contained within the *Item 2* list. For example, an unordered list of states and cities within those states could be marked up as

```
<ul>
   <li>California</li>
   <li>Oregon
      <ul>
         <li>Portland</li>
         <li>Salem</li>
      </ul>
   </li>
   <li>Washington</li>
</ul>
```

Most browsers will differentiate the various levels by increasing the indentation and using a different list symbol at each level of nested lists, for example, rendering the HTML code above as

- California
- Oregon
  - Portland
  - Salem
- Washington

The markers used at each level and the amount of indentation applied to each nested list is determined by style sheets, either those built into the browser or those supplied by the page designer. You'll explore this technique in Tutorial 2.

Description Lists

A third type of list is the **description list** containing a list of terms and matching descriptions. The description list is grouped by the `dl` (description list) element, the terms are marked with the `dt` (description term) element, and the description(s) associated with each term is marked by the `dd` element. The general structure is

```
<dl>
   <dt>term1</dt>
   <dd>description1</dd>
   <dt>term2</dt>
   <dd>description2a</dd>
   <dd>description2b</dd>
   ...
</dl>
```

where *term1*, *term2*, and so forth are the terms in the list, and *description1*, *description2a*, *description2b*, and so forth are the descriptions associated with the terms. Note that descriptions must always directly follow the term they describe and that more than one description may be provided with each term.

By default, most browsers indent the descriptions associated with each term. Markers are rarely displayed alongside either the description term or the description.

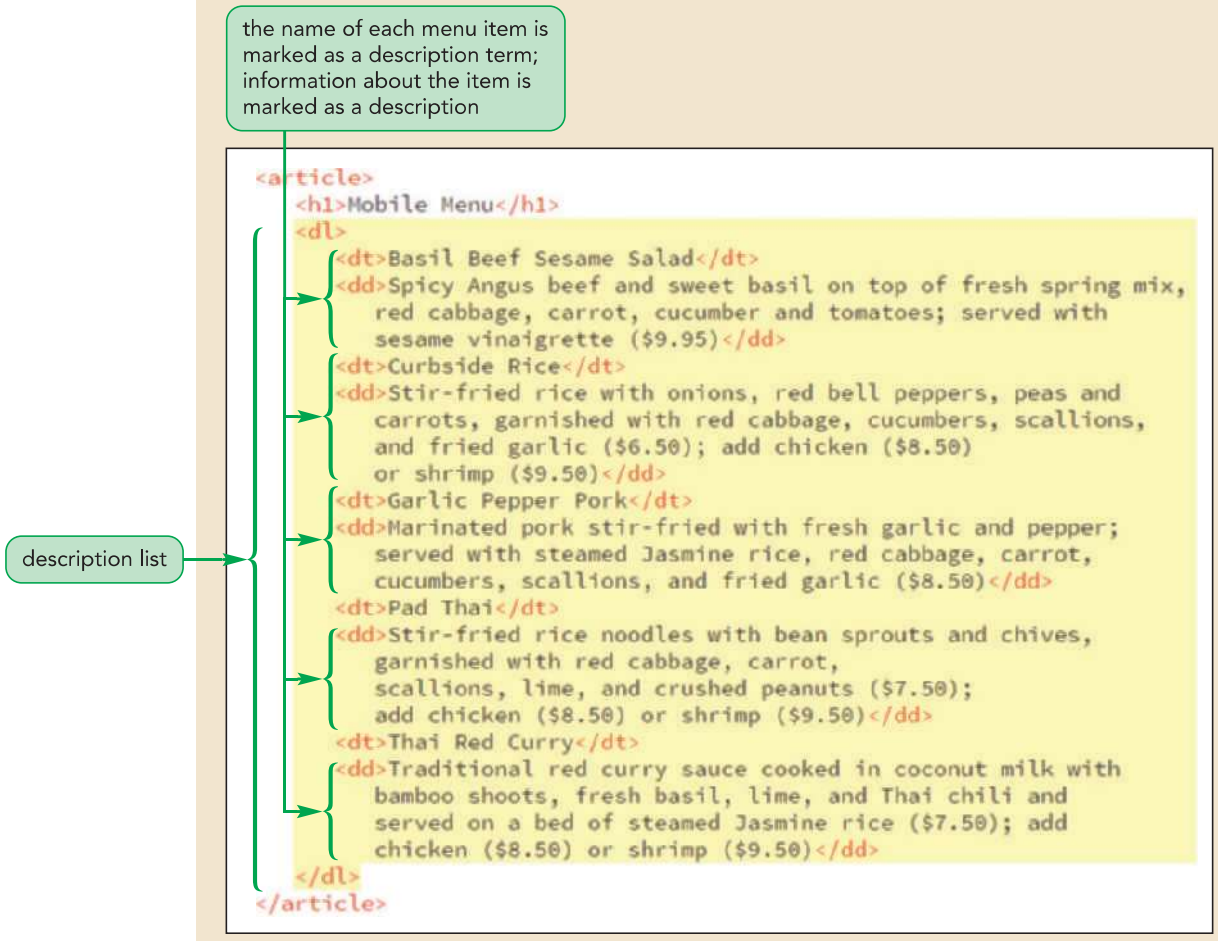
Sajja wants to use a description list in a page that displays some of the menu items sold by Curbside Thai. He's already started work on the HTML code but needs you to complete it by adding the markup for the description list.

To complete the menu page:

1. Open the **ct_menu_txt.html** file from the html01 ▸ tutorial folder in your HTML editor. Enter ***your name*** and ***the date*** in the comment section and save the file as ct_menu.html.
2. Open the **ct_pages.txt** file in your text editor if it is not already open and copy the five menu items listed in the Mobile Menu section.
3. Return to the **ct_menu.html** file in your HTML editor and paste the copied text directly after the `<h1>Mobile Menu</h1>` tag.
4. Enclose the entire menu within an opening and closing **`<dl>`** tag.
5. Mark the name of each menu item using the **`dt`** element. Mark the corresponding description using the **`dd`** element. Indent your code to make it easier to read and interpret.

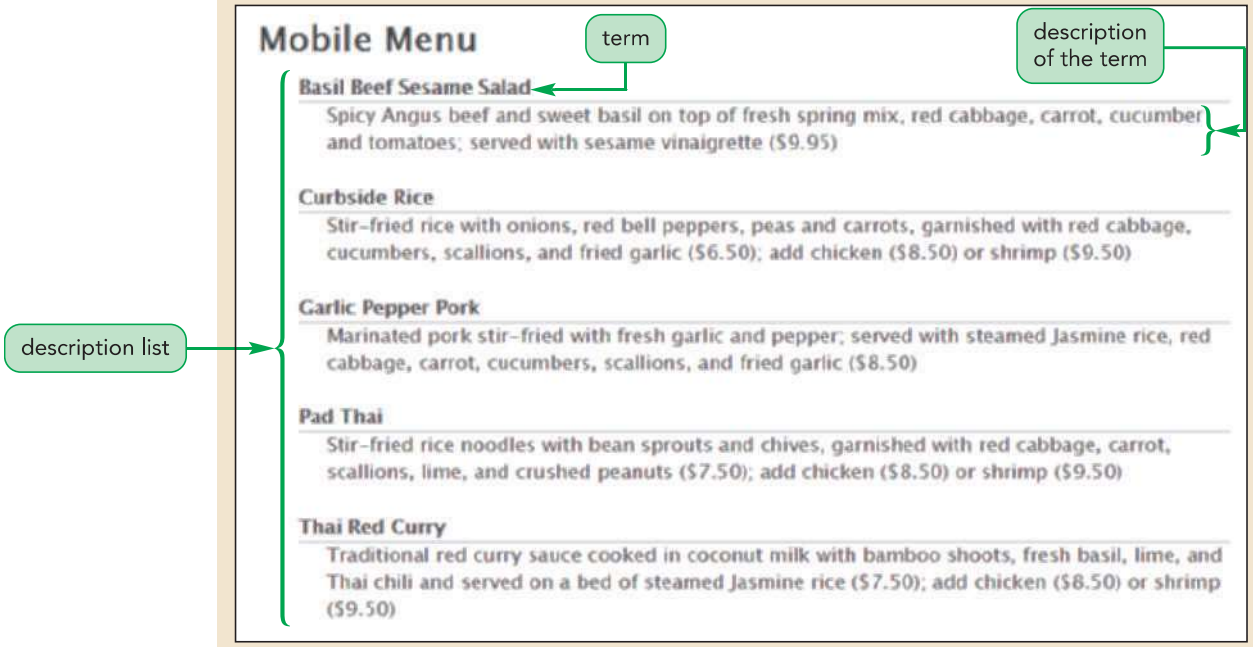
   Figure 1–32 shows the completed code for the description list of the mobile menu.

Figure 1–32 Marking the restaurant menu as a description list



6. Save your changes to the file and then open the **ct_menu.html** file in your browser. Figure 1–33 shows the completed menu for Curbside Thai.

Figure 1–33 Curbside Thai menu as a description list



Note that the style sheet that Sajja uses for his website inserts a dividing line between each term and description in the list. This is *not* the default browser style for description lists.

Description lists can be used with any general list that pairs one list of items with another list providing additional information. For example, Sajja has a page that lists the times and locations at which the Curbside Thai truck will make an appearance. Complete this page by enclosing the content within a description list, marking the times as the list "terms" and the locations as the list "descriptions".

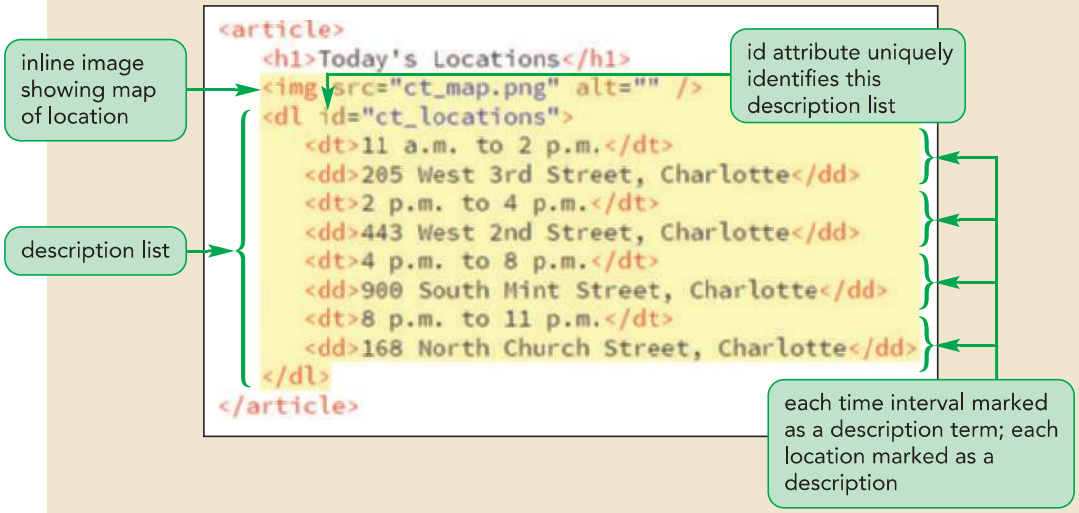
To create a page of times and locations:

1. Open the **ct_locations_txt.html** file from the html01 ▸ tutorial folder in your HTML editor. Enter ***your name*** and ***the date*** in the comment section and save the file as **ct_locations.html**.
2. Return to **the ct_pages.txt** file in your text editor and copy the four locations from the Today's Locations section.
3. Return to the **ct_locations.html** file in your HTML editor and paste the copied text directly after the `<h1>Today's Locations</h1>` tag.
4. Mark the entire list of times and locations using the `dl` element. Mark each time using the `dt` element and each location using the `dd` element. Indent your code to make it easier to read and interpret.
5. In order to distinguish this description list from other description lists in the website, add the attribute `id="ct_locations"` to the opening `<dl>` tag.
6. Sajja has a map that he wants displayed alongside the list of times and locations. Directly after the `h1` element within the `article` element, insert the following inline image:

```
<img src="ct_map.png" alt="" />
```

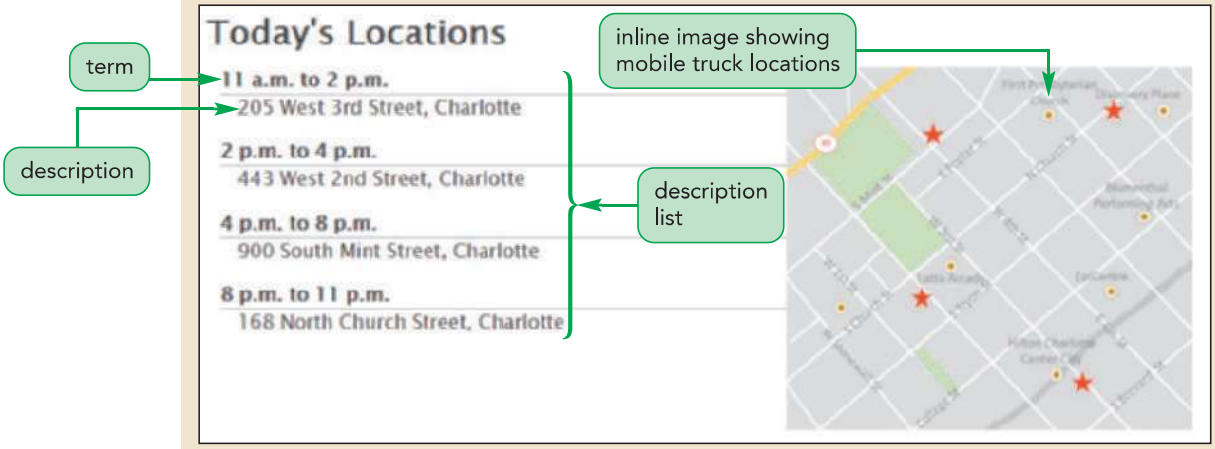
Figure 1–34 highlights the newly added code for the Today's Locations page.

Figure 1–34 Creating a description list



7. Save your changes to the file and then open the **ct_locations.html** file in your browser. Figure 1–35 shows the appearance of the page. Remember, the placement of items on the screen is a result of the style sheets.

Figure 1–35 Locations of the Curbside Thai food truck



From this page, Curbside Thai customers can quickly find the mobile truck. A page like this will have to be updated, probably daily, as the truck moves around. This is often better accomplished using database programs on the web server that will generate code for both the HTML and the inline image file.

INSIGHT

Marking Dates and Times

The adage that nothing ever quite disappears on the Internet also means that the web is populated with old articles, documents, and news stories that are no longer relevant or perhaps, even accurate. Any content you publish to the web should be time-stamped to document its history. One way of marking a date-time value is with the following `time` element

```
<time datetime="value">content</time>
```

where *value* is the date and time associated with the enclosed content. Dates should be entered in the *yyyy-mm-dd* format, where *yyyy* is the four-digit year value, *mm* is the two-digit month value, and *dd* is the two-digit day value. Times should be entered in the *hh:mm* format for the two-digit hour and minute values entered in 24-hour time. To combine both dates and times, enter the date and time values separated by a space or the letter `T` as in the following code:

```
<footer>Last updated at:
   <time datetime="2021-03-01T14:52">March 1 2021 at 2:52
p.m.</time>
</footer>
```

For international applications, you can base your time values on the common standard of Greenwich Mean Time. For example, the following code includes the information that the time is based on the Eastern time zone, which is 5 hours behind Greenwich Mean Time:

```
<p>Webinar starts at:
   <time datetime="2021-03-10T20:30-05:00">3:30 p.m.
(EST)</time>
</p>
```

While the value of the `datetime` attribute is not visible to users, it is readable by machines such as search engines, which can include the date and time in reporting search results. You can read more about the `time` element on the W3C website, including information on marking a time duration between two events.

You've now created six web pages for the Curbside Thai website. Next, you'll link these pages together so that users can easily navigate between the pages in the website. You'll start by creating a navigation list.

Navigation Lists

A **navigation list** is an unordered list of hypertext links placed within the nav element. The general structure is

```
<nav>
   <ul>
      <li>link1</li>
      <li>link2</li>
…
   </ul>
</nav>
```

where *link1*, *link2*, and so forth are hypertext links. While hypertext links can be placed anywhere within the page, having a central list of links makes the website easier to work with and navigate.

Add this structure to the About Curbside Thai web page, creating entries for each of the six web pages you created in this tutorial.

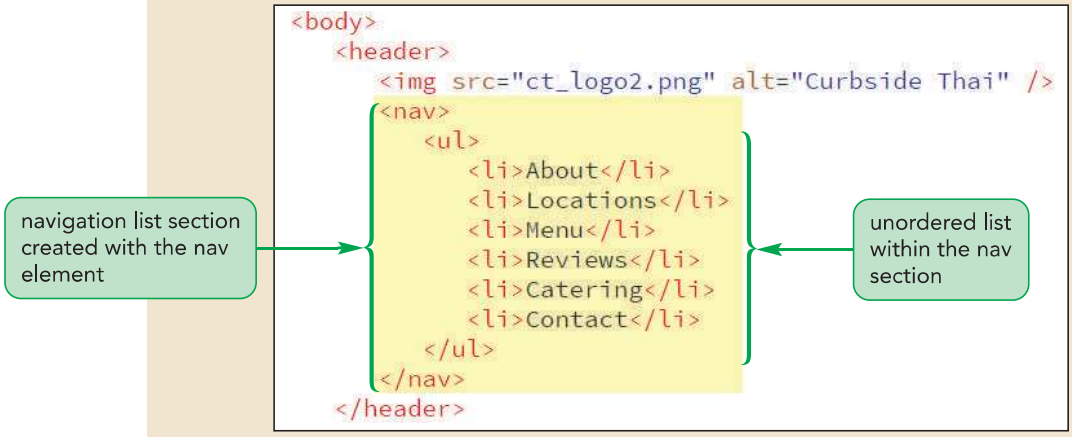
To create a navigation list:

1. Open the **ct_about.html** file in your HTML editor if it is not already open.
2. Go to the body header and, directly below the inline image for the Curbside Thai logo, insert the following navigation list:

```
<nav>
   <ul>
      <li>About</li>
      <li>Locations</li>
      <li>Menu</li>
      <li>Reviews</li>
      <li>Catering</li>
      <li>Contact</li>
   </ul>
</nav>
```

Figure 1–36 highlights the structure of the navigation list.

Figure 1–36 Creating a navigation list



3. Save your changes to the file and then reopen the **ct_about.html** file in your browser.

Figure 1–37 shows appearance of the navigation list.

Figure 1–37 Navigation list for the Curbside Thai website

Note that the appearance of the navigation list in the ct_about.html file is based on styles in Sajja's style sheets. Navigation lists can be displayed in a wide variety of ways depending on the styles being employed and the same navigation list might be laid out horizontally for desktop devices and vertically for mobile devices. You'll learn more about how to format navigation lists in Tutorial 5.

Now that you've created the structure of the navigation list, mark the list items as hypertext links.

Working with Hypertext Links

Hypertext is created by enclosing content within a set of opening and closing `<a>` tags in the following structure

```
<a href="url">content</a>
```

TIP

Keep your filenames short and descriptive so that users are less apt to make a typing error when accessing your website.

where *url* is the **Uniform Resource Locator** (**URL**), which is a standard address format used to link to a variety of resources including documents, email addresses, telephone numbers, and text messaging services, and *content* is the document content marked as a link. When linking to another HTML file in the same folder, the URL is simply the name of the file. For example, a hypertext link to the ct_menu.html file would be marked as

```
<a href="ct_menu.html">Menu</a>
```

When the user clicks or touches the word *Menu*, the browser will load the ct_menu.html file in the browser. Note that filenames are case sensitive on some web servers so that those servers differentiate between files named ct_menu.html and CT_Menu.html. The standard for all web filenames is to always use lowercase letters and to avoid using special characters and blank spaces.

The default browser style is to underline hypertext links and to display those links in a different text color if the user has previously visited the page. However, page designers can substitute different hypertext link styles from their own style sheets. We'll explore this technique in Tutorial 2.

REFERENCE

Marking a Hypertext Link

- To mark content as a hypertext link, use

```
<a href="url">content</a>
```

where *url* is the address of the linked document, and *content* is the document content that is being marked as a link.

Mark the six entries in the navigation list, pointing each entry to the corresponding Curbside Thai page.

To create hypertext links:

1. Return to the **ct_about.html** file in your HTML editor.
2. Mark the first entry as a hypertext link pointing to ct_about.html file by changing the list item to

```
<a href="ct_about.html">About</a>
```

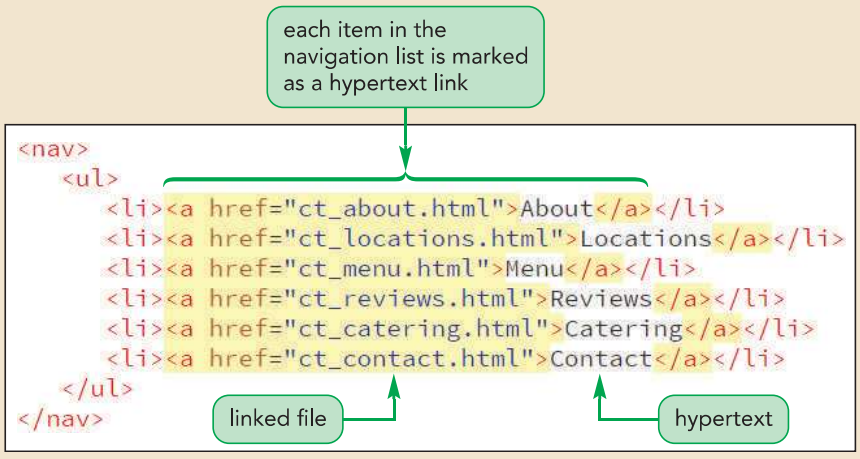
3. Change the code of the second list item to

```
<a href="ct_locations.html">Locations</a>
```

4. Continuing in the same fashion, change the Menu entry to a link pointing to the **ct_menu.html** file, the Reviews entry to a link pointing to the **ct_reviews.html** file, the Catering entry to a link pointing to the **ct_catering.html** file, and the Contact entry to a link pointing to the **ct_contact.html** file.

   Figure 1–38 highlights the newly added code that changes all of the items in the navigation list to hypertext links.

Figure 1–38 Marking hypertext links



5. Save your changes to the file and then reopen the ct_about.html file in your browser.
6. Click each of the six navigation list entries and verify that the browser loads the corresponding web page. Use the Back button on your browser to return to the About Curbside Thai page after you view each document.

Trouble? If the links do not work, be sure your code matches Figure 1–38. For example, check the spelling of each filename in the href attribute of each <a> tag to ensure it matches the filename of the corresponding Curbside Thai web page and check to be sure you have all needed opening and closing tags.

You may have noticed that when your mouse pointer moved over a hypertext link in the navigation list, the appearance of the link changed to white text on a black background. This is an example of a **rollover effect**, which is used to provide visual clues that the text is hypertext rather than normal text. You'll learn how to create rollover effects in Tutorial 2.

Turning an Inline Image into a Link

Inline images can also be turned into links by enclosing the image within opening and closing <a> tags. Turn the Curbside Thai logo into a hyperlink that points to the Startup page you opened in the first session.

To mark an image as a hypertext link:

1. Return to the **ct_about.html** file in your HTML editor.
2. Mark the image in the body header as a hyperlink by changing the HTML code for the inline image to

```
<a href="ct_start.html"><img src="ct_logo2.png"
alt="Curbside Thai" /></a>
```

Figure 1–39 highlights the code to change the logo image to a hypertext link.

Figure 1–39 Marking an inline image as a hypertext link

reference to the hypertext link

```
<body>
   <header>
      <a href="ct_start.html"><img src="ct_logo2.png" alt="Curbside Thai" /></a>
      <nav>
         <ul>
```

3. Save your changes to the file and then reopen the ct_about.html file in your browser.
4. Click the Curbside Thai logo and verify that the browser opens the Curbside Thai Startup page. Click the Back button to return to the About Curbside Thai page.

Sajja wants to be able to jump to any document in the Curbside Thai website from any page. He asks you to copy the hypertext links, including the image hyperlink, you just created in the ct_about.html file to the other documents in the website.

To copy and paste the hypertext links:

1. Return to the **ct_about.html** file in your HTML editor.
2. Copy the entire content of the page header from the opening `<header>` tag through to the closing `</header>` tag, including the revised code for the company logo and navigation list.
3. Go to the **ct_locations.html** file in your HTML editor. Paste the copied HTML code, replacing the previous page header in this document. Save your changes to the file.
4. Repeat the previous step for the **ct_menu.html**, **ct_reviews.html**, **ct_catering.html**, and **ct_contact.html** files, replacing the body header in each of those documents with the revised header from ct_about.html. Save your changes to each file.
5. Reopen the **ct_locations.html** file in your browser and verify that you can jump from one page to another by clicking items in the navigation list at the top of each page. Also verify that you can jump to the Startup page at any time by clicking the Curbside Thai logo.

TIP

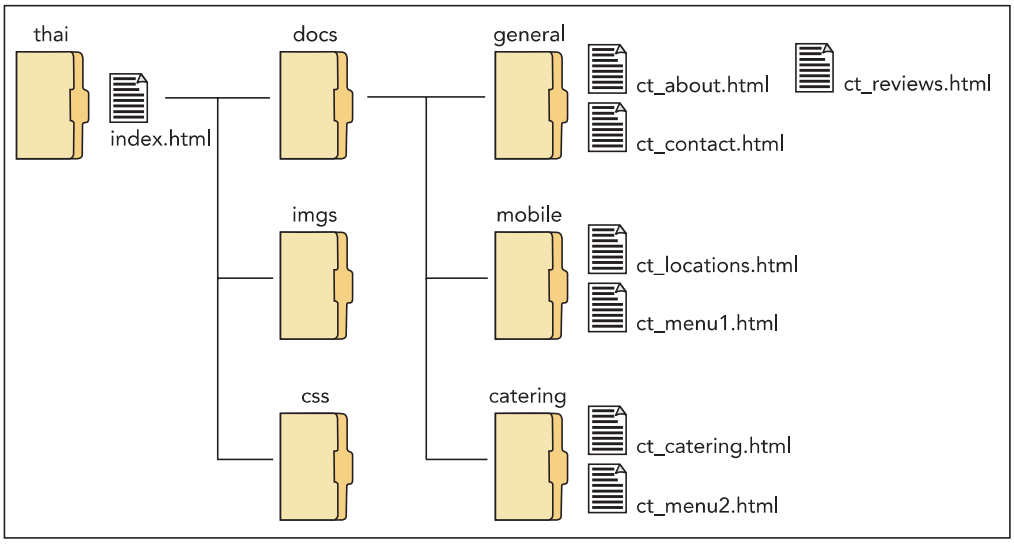
You can give your websites a uniform design by including the same navigation list on each page so that users can easily move from one page to the next.

Specifying the Folder Path

In the links you created, the browser assumed that the linked files were in the same folder as the current page. However, large websites containing hundreds of documents often place documents in separate folders to make them easier to manage.

Figure 1–40 shows a preview of how Sajja might organize his files as the Curbside Thai website increases in size and complexity. In this structure, all folders start from a **root folder** named *thai* that contains the site's home page, which Sajja has stored in the index.html file. Sajja has moved all of his images and CSS style sheet files into their own folders. He has divided the rest of the web pages among three subfolders: the general folder for pages containing general information about the restaurant, the mobile folder for pages with content specifically about the mobile food service, and the catering folder for pages describing Curbside Thai's catering opportunities.

Figure 1–40 A sample folder structure



To create links between files in separate folders, you must provide a path to the linked file. HTML supports two kinds of paths: absolute and relative.

Absolute Paths

An **absolute path** is a path that starts from the root folder and processes down the entire folder structure described with the expression

```
/folder1/folder2/folder3/file
```

where *folder1* is the root folder, followed by the subfolders *folder2*, *folder3*, and so forth, down to the linked file. Based on the structure shown previously in Figure 1–40, an absolute path pointing to the ct_catering.html file would be

```
/thai/docs/catering/ct_catering.html
```

If files are located on different drives as well as in different folders, you must include the drive letter in the path with the expression

```
/drive|/folder1/folder2/folder3/file
```

where *drive* is the letter assigned to the drive. Note that the drive letter must be followed by the | character. Thus, if the ct_catering.html file were located on drive E, the absolute path that includes the drive would have the expression

```
/E|/thai/docs/catering/ct_catering.html
```

Note that you don't have to include a drive letter if the linked document is located on the same drive as the current file.

Relative Paths

When many folders and subfolders are involved, absolute path expression can quickly become long and cumbersome to work with. For this reason, most web designers prefer **relative paths** in which the path is expressed relative to the location of the current document. If the current document and linked file are in the same folder, there is no path and you need only include the filename. If the linked file is in a subfolder of the current document, the path includes all of the subfolder names starting from the location of the current page using the expression

```
folder1/folder2/folder3/file
```

where *folder1*, *folder2*, *folder3*, and so forth are subfolders of the current document. The relative path to the ct_about.html file starting from the index.html file is

```
docs/general/ct_about.html
```

Note that relative paths are often expressed in terms of familial relationships such as parent, child, descendant, sibling, and so forth in order to indicate the hierarchical nature of the folder structure. Relative paths can also go up the hierarchy to parent folders by including the symbol (..), which means "go up one level." Thus, to go from ct_about.html in the general folder up two levels to the index.html file, enter the expression

```
../../index.html
```

TIP

You can reference the current folder using a single period (.) character.

Finally, to go sideways in the folder structure by going to a file in a different folder but on the same level, you go up to the parent folder and then back down to a different child folder. To go from the ct_about.html file in the general folder to the ct_locations.html file in the mobile folder, use the relative path expression

```
../mobile/ct_locations.html
```

In this expression, the link goes up to the parent folder docs through the use of the .. reference and then back down through the mobile folder to ct_locations.html.

You should almost always use relative paths in your links. If you have to move your files to a different computer or server, you can move the entire folder structure without having to edit the relative paths you've created. If you use absolute paths, you will have to revise each link to reflect the new location of the folder tree on the new device.

Setting the Base Path

A browser resolves relative paths based on the location of the current document. You define a different starting point for relative paths by adding the following `base` element to the document head

```
<base href="url" />
```

where *url* is the location that you want the browser to use when resolving relative paths in the current document. The `base` element is useful when a single document from the website is moved to a new folder. Rather than rewriting all of the relative paths to reflect the document's new location, the `base` element can point to the document's old location allowing relative paths to work as before.

PROSKILLS

Decision Making: Managing Your Website

Websites can quickly grow to dozens or hundreds of pages. As the size of a site increases, it becomes more difficult to get a clear picture of the site's structure and content. Imagine deleting or moving a file in a website that contains dozens of folders and hundreds of files. Could you easily project the effect of this change? Would all of your hypertext links still work after you moved or deleted the file?

To effectively manage a website, you should implement clear decision-making skills by following a few important rules. The first is to be consistent in how you structure the site. If you decide to collect all image files in one folder, you should continue that practice as you add more pages and images. Websites are more likely to break down if files and folders are scattered throughout the server without a consistent rule or pattern. Decide on a structure early and stick with it.

A second rule is to decide on and then create a folder structure that matches the structure of the website itself. If the pages can be easily categorized into different groups, those groupings should also be reflected in the groupings of the subfolders. The names you assign to your files and folders should also reflect their uses on the website. This makes it easier for you to predict how modifying a file or folder might impact other pages on the website.

Finally, you should document your work by adding comments to each new web page. Comments are useful not only for colleagues who may be working on the site but also for the author who must revisit those files months or even years after creating them. The comments should include

- The page's filename and location
- The page's author and the date the page was initially created
- A list of any supporting files used in the document, such as image and audio files
- A list of the files that link to the page and their locations
- A list of the files that the page links to and their locations

By following these rules, you can reduce a lot of the headaches associated with maintaining a large and complex website.

Linking to a Location within a Document

TIP

In general, a web page should not span more than one or two screen heights. Studies show that users often skip long pages where the content runs off the screen.

Hypertext can point to locations within a document. For example, you could link a specific definition within a long glossary page to save users the trouble of scrolling through the document. Websites containing the text of novels or plays can contain links to key passages or phrases within those works. When a link is established to a location within a document, the browser will jump to that location automatically scrolling the page to the linked location.

Marking Locations with the id Attribute

In order to enable users to jump to a specific location within a document, identify that location by adding the following `id` attribute to an element tag at that location

```
id="text"
```

where *text* is the name assigned to the ID. Imagine that Sajja writes a long page describing the full menu offered by Curbside Thai. He could mark the location in the page where the lunch menu is displayed by adding the following `id` attribute to the h2 heading that marks the start of the Lunch Menu section.

```
<h2 id="lunch">Lunch Menu</h2>
```

TIP

IDs are case-sensitive: an ID of "top" is different from an ID of "TOP".

Note that IDs must be unique. If you assign the same ID to more than one element, the browser will jump to the first occurrence of that ID value.

Linking to an id

Once you've marked the location with an ID, you link to that element using the following hypertext link:

```
<a href="file#id">content</a>
```

where *file* points to the location and filename of the linked document and *id* is associated with the element within that document. The following hypertext link points to the element with the ID "lunch" within the ct_fullmenus.html file.

```
<a href="ct_fullmenus.html#lunch">View our Lunch Menu</a>
```

To link to a location within the current page, include only the ID value along with the # symbol. Thus, the following hypertext link points to the lunch ID within the current web page:

```
<a href="#lunch">View our Lunch Menu</a>
```

In both cases, clicking or tapping the link will cause the browser to automatically scroll to the location within the page.

Anchors and the name Attribute

Early web pages did not support the use of the `id` attribute as a way of marking locations within a document. Instead, they used the `<a>` tag as an anchor to mark that page location (hence the "a" in `<a>` tag). The general form of the anchor was

```
<a name="anchor">content</a>
```

where *anchor* is the name given to the anchored text. Inserting content within the `<a>` tag was optional because the primary purpose of the tag was to mark a document location, not to mark up content. The following code would establish an anchor at the start of the lunch section in the Curbside Thai full menu:

```
<h2><a name="lunch"></a>Lunch Menu</h2>
```

Once an anchor had been set, you link to the anchor using the same syntax you would use with the `id` attribute. The use of anchors is a deprecated feature of HTML and is not supported in strict applications of XHTML, but you will still see anchors used in older websites.

REFERENCE

Linking to a Location within a Document

- To mark a location, add a unique ID to an element at that document location using the following `id` attribute

  ```
  id="text"
  ```

  where *text* is the value of the ID.
- To link to that location from a different document, use the hypertext reference

  ```
  <a href="file#text">content</a>
  ```

  where *file* is the name and path location (if necessary) of the external file and *text* is the value of the ID.
- To link to that location from within the same document, use the hypertext reference

  ```
  <a href="#text">content</a>
  ```

Linking to the Internet and Other Resources

The type of resource that a hypertext link points to is indicated by the link's URL. All URLs share the general structure

```
scheme:location
```

where *scheme* indicates the resource type and *location* provides the resource location. The name of the scheme is taken from the network protocol used to access the resource where a **protocol** is a set of rules defining how information is passed between two devices. Pages on the web use the **Hypertext Transfer Protocol** (**HTTP**) protocol and therefore the URL for many web pages start with the `http` scheme. Other schemes that can be included within a URL are described in Figure 1–41.

Figure 1–41 Commonly used URL schemes

Scheme	Description
`fax`	A fax phone number
`file`	A document stored locally on a user's computer
`ftp`	A document stored on an FTP server
`geo`	A geophysical coordinate
`http`	A resource on the World Wide Web
`https`	A resource on the World Wide Web accessed over a secure encrypted connection
`mailto`	An email address
`tel`	A telephone number
`sms`	A mobile text message sent via the Short Message Service

Linking to a Web Resource

If you have ever accessed the web, you should be very familiar with website URLs, which have the general structure

```
http://server/path/filename#id
```

or for secure connections

```
https://server/path/filename#id
```

where *server* is the name of the web server hosting the resource, *path* is the path to the file on that server, *filename* is the name of the file, and if necessary, *id* is the name of an id or anchor within the file. For example, the following URL uses the HTTP protocol to access the web server at *www.curbsidethai.com*, linking to the document location named *lunch* within the ct_menus.html file in the /thai/docs folder:

```
http://www.curbsidethai.com/thai/docs/ct_menus.html#lunch
```

URLs are often entered in a more abbreviated form, *http://www.curbsidethai.com* for example, with no path or filename. Those URLs point to the default home page located in the top folder in the server's folder tree. Many servers use index.html as the filename for the default home page, so the URL *http://www.curbsidethai.com* would be equivalent to *http://www.curbsidethai.com/index.html*.

INSIGHT

Understanding Domain Names

The server name portion of a URL is also called the **domain name**. By studying a domain name, you learn about the server hosting the website. Each domain name contains a hierarchy of names separated by periods (.), with the top level appearing at the far right end. The top level, called an **extension**, indicates the general audience supported by the web server. For example, *.edu* is the extension reserved for educational institutions, *.gov* is used for agencies of the United States government, and *.com* is used for commercial sites or general-use sites.

The next lower level appearing to the immediate left of the extension displays the name of the individual or organization hosting the site. The domain name *curbsidethai.com* indicates a commercial or general-use site owned by Curbside Thai. To avoid duplicating domain names, the top two levels of the domain must be registered with the **Internet Assigned Numbers Authority** (**IANA**) before they can be used. You can usually register your domain name through your web hosting company. Note: You must pay an annual fee to keep a domain name.

The lowest levels of the domain, which appear farthest to the left in the domain name, are assigned by the individual or company hosting the site. Large websites involving hundreds of pages typically divide their domain names into several levels. For example, a large company like Microsoft might have one domain name for file downloads—*downloads.microsoft.com*—and another domain name for customer service—*service.microsoft.com*. Finally, the first part of the domain name displays the name of the hard drive or resource storing the website files. Many companies have standardized on www as the initial part of their domain names.

Linking to an Email Address

Many websites use email to allow users to communicate with a site's owner, sales representative, or technical support staff. You can turn an email address into a hypertext link using the URL:

```
mailto:address
```

where *address* is the email address. Activating the link opens the user's email program with the email address automatically inserted into the To field of a new outgoing message. To create a hypertext link to the email address *s.adulet@example.com*, you could use the following URL:

```
mailto:s.adulet@example.com
```

TIP

To link to more than one email address, add the addresses to the mailto link in a comma-separated list.

The mailto protocol also allows you to insert additional fields into the email message using the URL:

```
mailto:address?field1=value1&field2=value2&...
```

where *field1*, *field2*, and so forth are different email fields and *value1*, *value2*, and so forth are the field values. Fields include subject for the subject line of the email message and body for the message body. To create a link to an email message with the following content

```
TO: s.adulet@example.com
SUBJECT: Test
BODY: Test Message
```

you would use the URL

```
mailto:s.adulet@example.com?subject=Test&body=Test%20Message
```

Notice that the body text uses `%20` character code to represent a blank space since URLs cannot contain blank spaces.

On the Contact Us page, Sajja has inserted the Curbside Thai's email address. Convert this email address into a hypertext link.

To link to an email address:

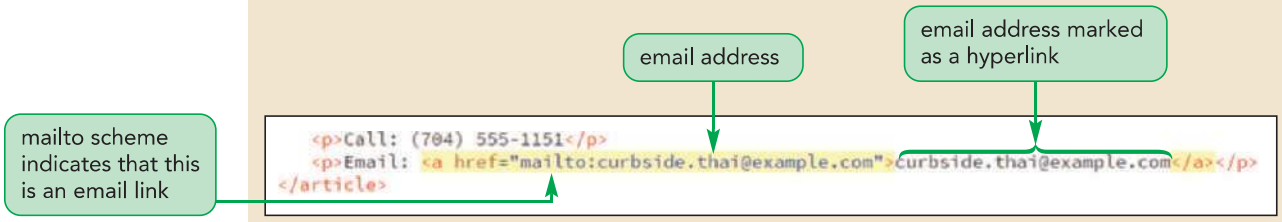
1. Go to the **ct_contact.html** file in your HTML editor.
2. Change the Curbside Thai email address into the following mailto hypertext link:

A mailto hypertext link to an external resource must include the mailto scheme name in order to be recognized by the browser.

```
<a href="mailto:curbside.thai@example.com">
   curbside.thai@example.com
</a>
```

Note that this is a fictional email address. If you want to test this link, change the URL to a link pointing to your own email address. Figure 1–42 highlights the hypertext code to the linked email address.

Figure 1–42 Linking to an email address



3. Save your changes to the file and then reopen the ct_contact.html file in your browser.
4. Click the email address link and verify that your device opens your email program with the Curbside Thai address already entered. Close the email program without sending a message.

Trouble? Depending on your device, you may have to set up your email program to accept hypertext links.

INSIGHT

Email Links and Spam

Use caution when adding email links to your website. While it may make it more convenient for users to contact you, it also might make you more vulnerable to spam. **Spam** is unsolicited email sent to large numbers of people, promoting products, services, and in some cases inappropriate websites. Spammers create their email lists by scanning discussion groups, stealing Internet mailing lists, and using programs called **email harvesters** to scan HTML code for the email addresses contained in mailto URLs. Many developers have removed email links from their websites in order to foil these harvesters, replacing the links with web forms that submit email requests to a secure server.

Fighting spammers is an ongoing battle, and they have proved very resourceful in overcoming some of the defenses people have created. As you develop your website, you should carefully consider how to handle email addresses and review the most current methods for safeguarding that information.

Linking to a Phone Number

With the increased use of mobile phones to access the web, many developers now include links to phone numbers for their company's customer service or help line. Activating the link brings up the user's phone app with the number already entered, making it easier and more convenient to call the business or organization. The URL for a phone link is

```
tel:phone
```

where *phone* are the digits of the linked number. For example, the following code creates a telephone link to the Curbside Thai number:

```
Call: <a href="tel:+17045551151">(704) 555-1151</a>
```

TIP

Skype on the desktop uses `callto:` in place of the `tel:` scheme for telephone links. There are program scripts available on the web that you can use in order to work with both protocols.

Because websites are international, any telephone link should include the international dialing prefix (+1 for the United States) and the area code. Spaces or dashes between digits are optional with the exception of the + symbol before the international calling code. However, you can insert pauses in the phone number (used when accessing an extension) by inserting the `p` symbol, as in the following telephone link:

```
<a href="tel:+17045551151p22">Call: 555-1151 ext. 22</a>
```

Sajja asks you to change the telephone number from the Contact Us page into a telephone link.

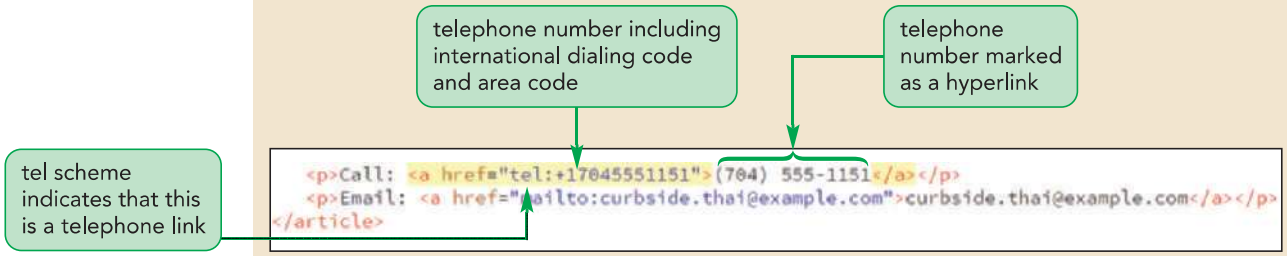
To link to a phone number:

1. Return to the **ct_contact.html** file in your HTML editor.
2. Change the Curbside Thai phone number into the following hypertext link:

```
<a href="tel:+17045551151">
   (704) 555-1151
</a>
```

Once again this number is fictional; you can change the URL to a link pointing to your own phone number if you want to test the link on a mobile device. Figure 1–43 highlights the hypertext code of the telephone link.

Figure 1–43 Marking a telephone link



3. Save your changes to the file.

HTML supports links to other types of telephony devices. You can create a link to a fax machine using the `fax:` scheme and a link to your text messaging app by using the `sms:` scheme.

Working with Hypertext Attributes

HTML provides several attributes to the a element that control the behavior and appearance of hypertext links. Figure 1–44 describes these attributes.

Figure 1–44 Attributes of the a element

Attribute	Description
`href="url"`	Provides the *url* of the hypertext link
`target=(_blank|_parent|_self|_top)`	Specifies where to open the linked document
`download="filename"`	Indicates that the link should be downloaded as a file, where *filename* is the name given to the downloaded file
`rel="type"`	Provides the relationship between the linked document and the current page
`hreflang="lang"`	Indicates the language of the linked document
`type="mime-type"`	Indicates the media type of the linked document

Using the `target` attribute, you can control how a page is opened. By default the target of a link replaces the contents of the current page in the browser window. In some websites, you will want to open a link in a new browser window or tab so that you can keep the current page and the linked page in view. To force a document to appear in a new window or tab, add the following `target` attribute to the `<a>` tag:

```
<a href="url" target="window">content</a>
```

where *window* is a name assigned to the browser window or browser tab in which the linked page will appear. You can choose any name you wish for the browser window or you can use one of the following target names:

- `_self` opens the page in the current window or tab (the default)
- `_blank` opens the page in a new unnamed window or tab, depending on how the browser is configured
- `_parent` opens the page in the parent of the current frame (for framed websites)
- `_top` opens the page in the top frame (for framed websites)

You should use the `target` attribute sparingly in your website. Creating secondary windows can clutter up a user's desktop. Also, because the page is placed in a new window, users cannot use the Back button to return to the previous page in that window; they must click the browser's program button or the tab for the original website. This confuses some users and annoys others. Many designers now advocate not using the `target` attribute at all, but instead provide the user with the choice of opening a link in a new tab or window.

PROSKILLS

Written Communication: Creating Effective Hypertext Links

To make it easier for users to navigate your website, the text of your hypertext links should tell readers exactly what type of document the link points to. For example, the link text

Click here for more information.

doesn't tell the user what type of document will appear when here is clicked. In place of phrases like "click here", you should use descriptive link text such as

For more information, view our list of frequently asked questions.

If the link points to a non-HTML file, such as a PDF document, include that information in the link text. If the linked document is extremely large and will take a while to download to the user's computer, include that information in your link text so that users can decide whether or not to initiate the transfer. For example, the following link text informs users of both the type of document and its size so users have this information before they initiate the link:

Download our complete manual (PDF 2 MB).

Finally, when designing the style of your website, make your links easy to recognize. Users should never be confused about a link. Also, if you apply a color to your text, do not choose colors that make your hyperlinks harder to pick out against the web page background.

Validating Your Website

After finishing the code for your website, you can validate that code to ensure that there are no syntax errors. While browsers are very forgiving of syntax errors and will often render the page correctly, you should still perform a validation test for those browsers that might not be so accommodating.

Many HTML editors and web content management systems have built-in validators. If you don't have direct access to an HTML validator you can upload your code to the validator at the W3C website. To see how a validator can catch HTML syntax errors, you will introduce errors to the ct_about.html file and then test that file in the W3C validator.

To introduce errors to the ct_about.html file:

1. Return to the **ct_about.html** file in your editor.
2. Scroll down and delete the line `<h1>About Us</h1>`.
3. Go to the `<img>` tag for the ct_photo1.png inline image and delete the alt attribute, `alt=""`, changing the tag to simply `<img src="ct_photo.png" />`.
4. Save your changes to the file.

Using the validator at the W3C website or one built into your editor, validate the ct_about.html file to see the impact of these changes.

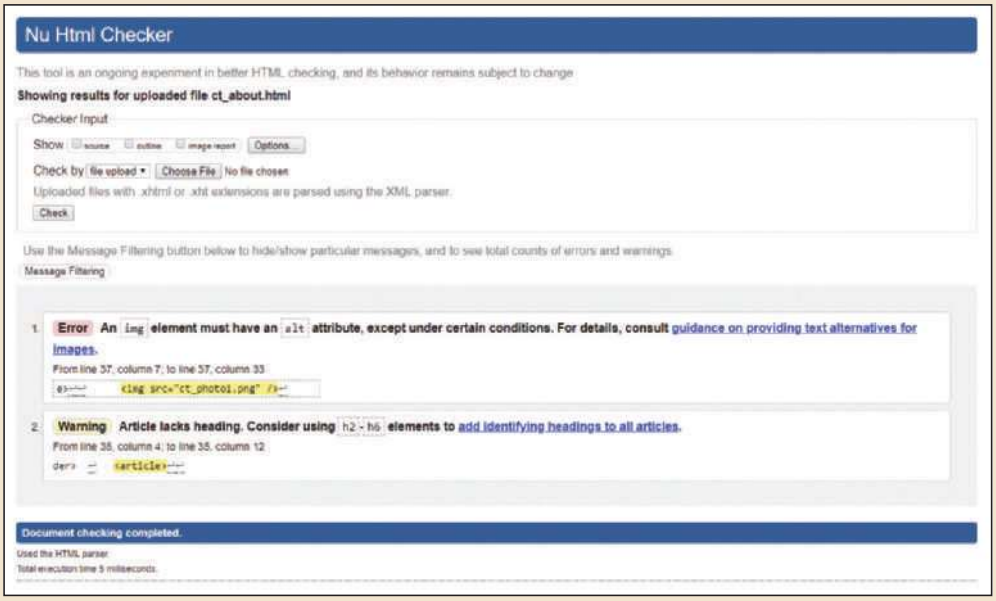
To validate the code in the ct_about.html file:

1. Open your browser to W3C validator at *https://validator.w3.org/*.

   Trouble? If you are using an HTML editor with a built-in validator, talk to your instructor about accessing that editor's validation command.
2. Click the **Validate by File Upload** tab within the web page.
3. Click the **Choose File** or **Browse** button in the web form and locate the **ct_about.html** file from the html01 ▸ tutorial folder.
4. Click the **Check** button to validate the file.

   The validator reports 1 error and provides 1 warning. See Figure 1–45.

Figure 1–45 Validation report



Because every `img` element must have an `alt` attribute (to make the page accessible to *all users*), the validator reports an error that the inline image for the ct_photo.png image is missing alternate text. The validator also issues a warning that the page article is lacking a heading. While this is not a syntax error (hence the validator only issued a warning) it is strongly recommended that all articles have at least one heading.

To fix the errors in the ct_about.html file:

1. Return to the **ct_about.html** file in your editor.
2. Re-insert the heading **`<h1>About Us</h1>`** directly below the opening `<article>` tag.
3. Add the attribute `alt=""` to the img element for the ct_photo1.png inline image.
4. Save your changes to the file.
5. Return in your browser to the W3C validator at *https://validator.w3.org/* or run the validator within your HTML editor.
6. Retest the **ct_about.html** file, verifying that no errors or warnings are reported.

You've completed your work on the Curbside Thai website. Sajja will study your work and get back to you with future projects for his restaurant. For now, you can close any open files or applications.

REVIEW

Session 1.3 Quick Check

1. Which of the follow tags is *not* used to mark a list?
   a. `<ol> …</ol>`
   b. `<ul> … </ul>`
   c. `<dl> … </dl>`
   d. `<li> … </li>`
2. To have an ordered list count down from 100, which attributes should you use?
   a. `down start="100"`
   b. `reversed start="100"`
   c. `decrease start="100"`
   d. `reversed from="100"`
3. Lists of hypertext links should be enclosed within what tags?
   a. `<a> …</a>`
   b. `<dl> … </dl>`
   c. `<links> … </link>`
   d. `<nav> … </nav>`
4. To link text to the website *https://ww.mobilepanini.com*, use the tag:
   a. `<a link="https://www.mobilepanini.com"> … </a>`
   b. `<link a=" https://www.mobilepanini.com"> … </link>`
   c. `<a href=" https://www.mobilepanini.com"> … </a>`
   d. `<a src=" https://www.mobilepanini.com"> … </a>`
5. Using Figure 1–40, the relative path going from the ct_about.html file to the ct_catering.html file is:
   a. `../catering/ct_catering.html`
   b. `ct_catering.html`
   c. `/catering/ct_catering.html`
   d. `./catering/ct_catering.html`
6. What tag can be used to define the starting point for relative paths?
   a. `<start />`
   b. `<base />`
   c. `<a> … </a>`
   d. `<link />`
7. What tag should you use to mark the h1 heading with the location topHeading?
   a. `<h1 id="TopHeading">Mobile Panini</h1>`
   b. `<h1 id="topHeading">Mobile Panini</h1>`
   c. `<a href="topHeading"></a><h1>Mobile Panini</h1>`
   d. All of the above
8. To link to the email address sajja@example.com, use the URL:
   a. mail:sajja@example.com
   b. sms:sajja@example.com
   c. fax:sajja@example.com
   d. mailto:sajja@example.com
9. To link to the phone number 970-555-0002, use the URL:
   a. tel:9705550002
   b. phone:9705550002
   c. call:9705550002
   d. dial:9705550002

CODE

Coding Challenge 1

Data Files needed for this Coding Challenge: code1-1_txt.html

Use HTML to create a navigation list of 10 government websites as shown in Figure 1–46. Each item in the list should act as a hypertext link to the government side.

Figure 1–46 Coding Challenge 1-1 example page

The following are the URLs associated with each website:

- USA government (https://www.usa.gov)
- Library of Congress (https://www.loc.gov)
- U.S. Congress (https://www.congress.gov)
- Federal Depository Library System (https://www.fdlp.gov)
- Ben's Guide to US Government for Kids (https://bensguide.gpo.gov)
- Health Finder (https://www.healthfinder.gov)
- National Center for Health Statistics (https://www.cdc.gov/nchs)
- The Whitehouse (https://www.whitehouse.gov)
- U.S. Census Bureau (https://www.census.gov)
- CIA World Factbook (https://www.cia.gov/library/publications/the-world-factbook/)

Do the following:

1. Open the **code1-1_txt.html** file from the tutorial1 ▸ code1 folder. Enter ***your name*** and ***the date*** in the head section of the document and save the file as **code1-1.html** to the same location.
2. Within the body section of the file, enter the text **Government Sites on the Web**. Mark that text as an h1 heading.
3. Place the list of government websites within an unordered list nested within a navigation list. Mark each government website as a separate list item.
4. Mark the text of each government site as a hypertext link using the `<a>` tag. Set the `href` attribute to the URLs specified above.
5. Save your changes to the file.
6. Test the page in your browser, verifying that each item in the list is linked to the appropriate government website.
7. Submit the completed file to your instructor.

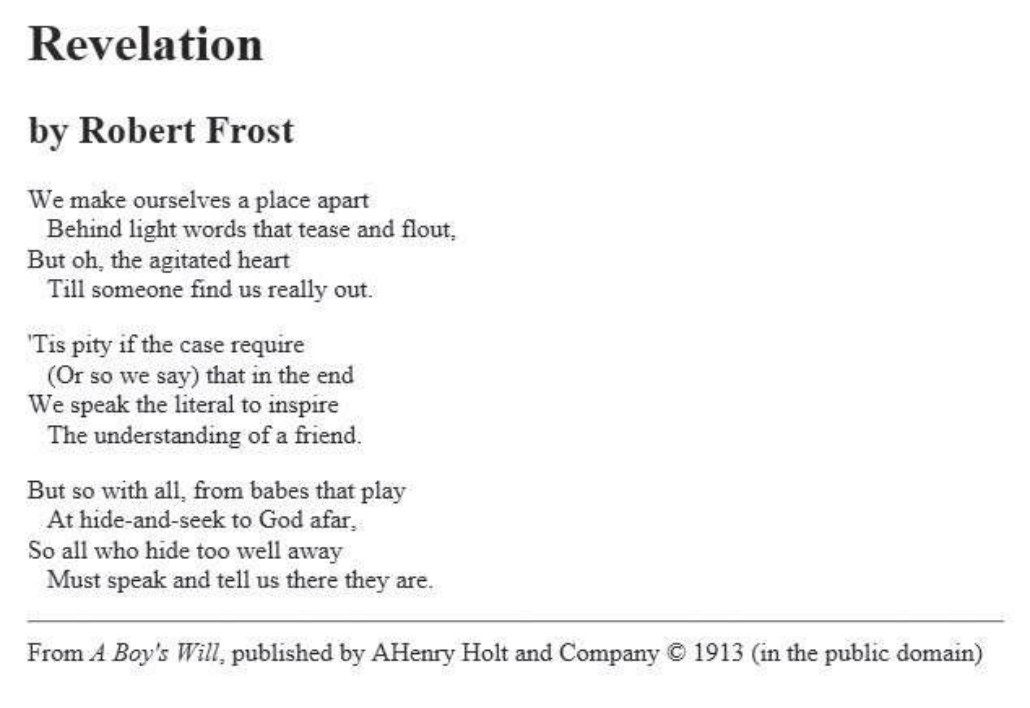
CODE

Coding Challenge 2

Data Files needed for this Coding Challenge: code1-2_txt.html

Mark up a web page containing the poem *Revelation* by Robert Frost shown in Figure 1–47.

Figure 1–47 Coding Challenge 1-2 example page



Revelation

by Robert Frost

We make ourselves a place apart
  Behind light words that tease and flout,
But oh, the agitated heart
  Till someone find us really out.

'Tis pity if the case require
  (Or so we say) that in the end
We speak the literal to inspire
  The understanding of a friend.

But so with all, from babes that play
  At hide-and-seek to God afar,
So all who hide too well away
  Must speak and tell us there they are.

From *A Boy's Will*, published by AHenry Holt and Company © 1913 (in the public domain)

Do the following:

1. Open the **code1-2_txt.html** file from the tutorial1 ▸ code2 folder. Enter ***your name*** and ***the date*** in the head section of the document and save the file as **code1-2.html** to the same location.
2. Mark the text *Revelation* as an h1 heading and the text *by Robert Frost* as an h2 heading.
3. Mark each stanza of the poem as a paragraph.
4. Within each of the three stanzas, end each line with a line break using the <br /> tag.
5. For alternating lines within the three stanzas, insert three   character entities to indent the text of those lines three spaces.
6. Replace the line of dashes after the poem with the <hr /> tag to indicate a horizontal rule.
7. Enclose the reference to the poem's origin within a <footer> tag.
8. Mark the text *A Boy's Will* using the <cite> tag.
9. Replace the word *copyright* with the character reference ©
10. Save your changes to the file.
11. View the page in your browser to verify that the poem's structure resembles that shown in Figure 1–47.
12. Submit the completed file to your instructor.

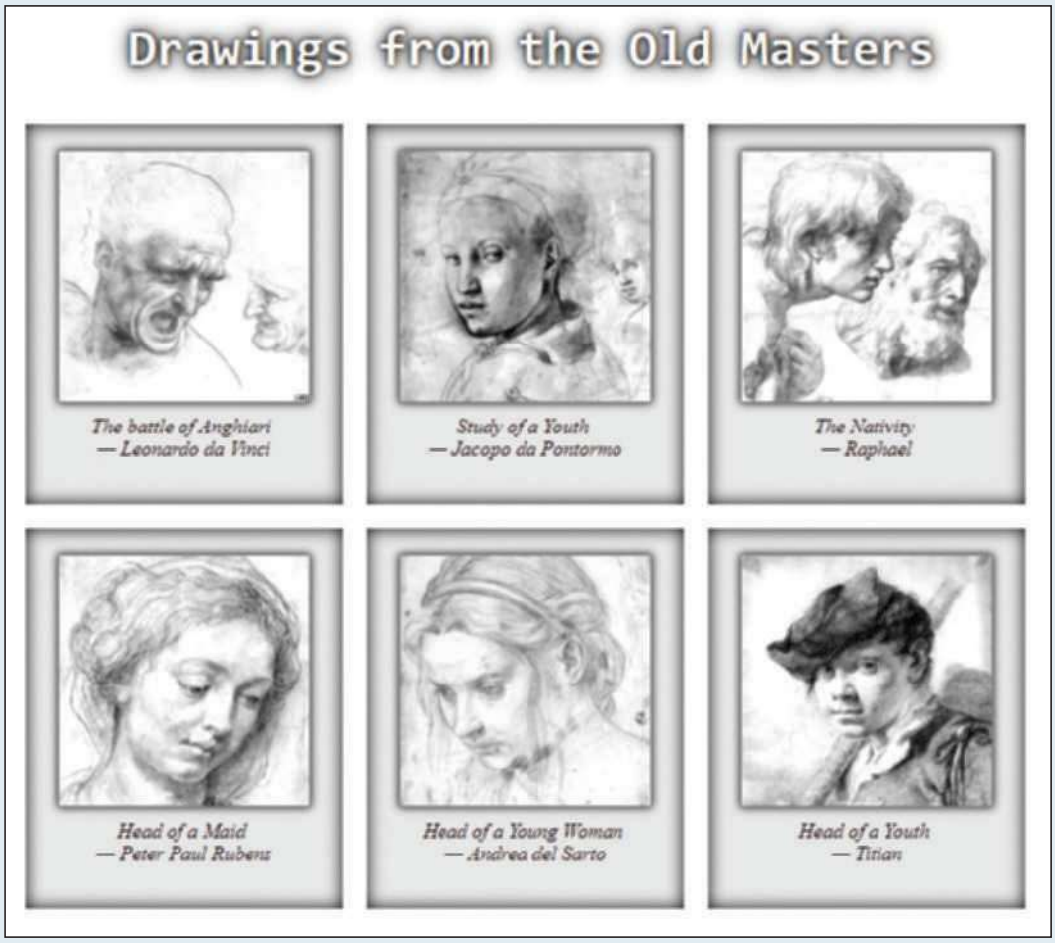
CODE

Coding Challenge 3

Data Files needed for this Coding Challenge: code1-3_txt.html, code1-3_styles.css, drawing01.png – drawing06.png

Display a gallery of six drawings by Renaissance masters using HTML and a CSS style sheet. The page containing the drawings is shown in Figure 1–48.

Figure 1–48 Coding Challenge 1-3 example page



Do the following:

1. Open the **code1-3_txt.html** file from the tutorial1 ▸ code3 folder. Enter ***your name*** and ***the date*** in the head section of the document and save the file as **code1-3.html** to the same location.
2. Add an attribute to the opening `<html>` tag to set the language of the page to "en" (for English).
3. Within the head section of the document do the following:
   a. Set the character encoding of the file to **utf-8**.
   b. Set the page title to **Coding Challenge 1-3**.
   c. Link the file to designs stored in the code1-3_styles.css style sheet file.
4. Within the page body, insert an h1 heading containing the text **Drawings from the Old Masters**.
5. After the h1 heading, insert six section elements using the `<section>` tag.

6. Within each of the six sections, insert an inline image using the `<img />` tag, displaying the image files drawing01.png through drawing06.png. For the six inline images, specify the following alternate text: **Leonardo da Vinci**, **Jacopo da Pontormo**, **Raphael**, **Peter Paul Rubens**, **Andrea del Sarto**, and **Titian**.
7. After each of the six inline images within the section elements, add a footer element using the `<footer>` tag. Within the six footers, add the following text captions: **The Battle of Anghiari --- Leonardo da Vinci**, **Study of a Youth -- Jacopo da Pontormo**, **The Nativity -- Raphael**, **Head of a Maid -- Peter Paul Rubens**, **Head of a Young Woman -- Andrea del Sarto**, and **Head of a Youth -- Titian**.
8. For the figure captions in the previous step, replace the six "--" character strings with a line break tag (`<br />`) followed by the — character reference.
9. Save your changes to the file.
10. View the page in your browser to verify that the drawing gallery resembles Figure 1–48.
11. Submit the completed file to your instructor.

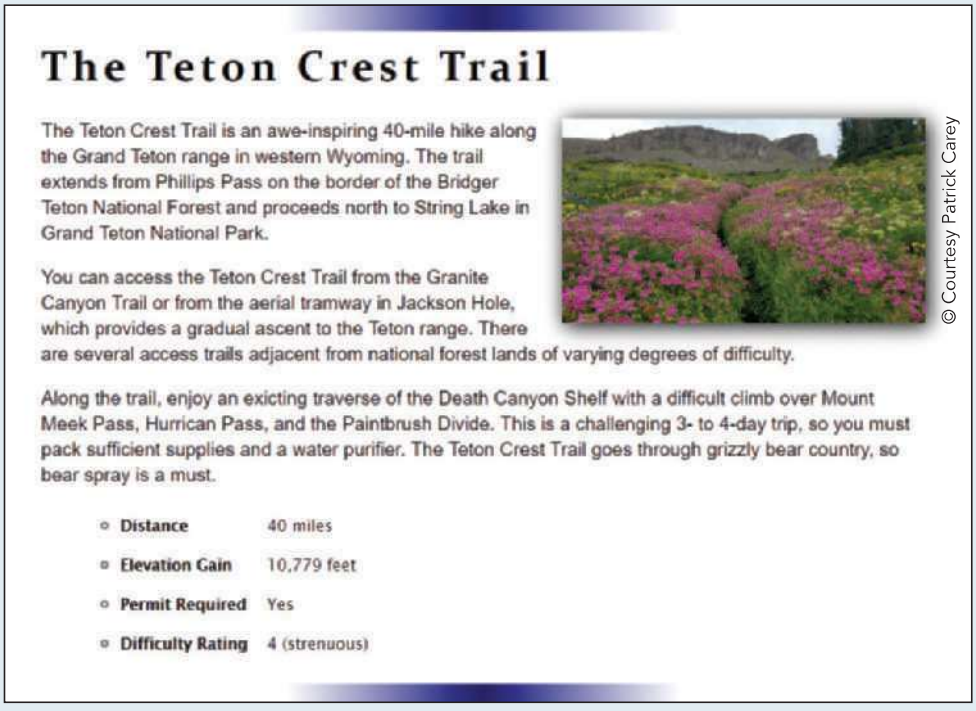
DEBUG

Coding Challenge 4

Data Files needed for this Coding Challenge: code1-4_txt.html, code1-4_styles.css, code1-4_image.png

You have been given a file containing several coding errors that need to be fixed for the page to pass validation. Locate and fix the errors in the web page. A preview of the fixed page is shown in Figure 1–49.

Figure 1–49 Coding Challenge 1-4 example page



Do the following:

1. Open the **code1-4_txt.html** file from the tutorial1 ▸ code3 folder. Enter ***your name*** and ***the date*** in the document and save the file as **code1-4.html** to the same location.
2. Test the page in the validator at the W3C website *(validator.w3.org)* or with another validator of your choice. Make a note of the errors reported as a guide to debugging the page. Not every error will be initially reported. There are 10 syntax errors in the HTML file.
3. Fix all 10 errors you discover and then resubmit the saved document to validation until it passes with no warnings or errors.
4. View the page in your browser to verify that the page resembles Figure 1–49.
5. Submit the completed file to your instructor.

PRACTICE

Review Assignments

Data Files needed for the Review Assignments: mp_index_txt.html, mp_menu_txt.html, mp_events_txt.html, mp_catering_txt.html, 2 CSS files, 2 PNG files, 1 TXT file

Curbside Thai has partnered with another food truck vendor Mobile Panini. Sajja asks you to create a website for the company similar to what you did for his restaurant. The site will have a home page, an online menu, a description of catering opportunities, and a calendar of upcoming events that Mobile Panini will host. A preview of the home page is shown in Figure 1–50.

Figure 1–50 Mobile Panini home page

The page text has already been written for you and style sheets and graphic files have been created. Your job will be to complete this project by writing the HTML markup.

Complete the following:

1. Use your HTML editor to open the **mp_index_txt.html**, **mp_menu.txt.html**, **mp_events_txt.html**, and **mp_catering_txt.html** files from the html01 ▸ review folder. Enter ***your name*** and ***the date*** in the comment section of each file, and save them as **mp_index.html**, **mp_menu.html**, **mp_events.html**, and **mp_catering.html** respectively.
2. Go to the **mp_index.html** file in your HTML editor. Within the document head, do the following:
   a. Use the `meta` element to set the character encoding of the file to **utf-8**.
   b. Add the following search keywords to the document: **Italian**, **Mobile**, **food**, and **Charlotte**.
   c. Set the title of the document to **Mobile Panini**.
   d. Link the document to the mp_base.css and mp_layout.css style sheet files.
3. Go to the document body and insert a `header` element containing the following:
   a. An inline image from the mp_logo.png file with the alternate text **Mobile Panini**. Mark the image as a hypertext link pointing to the mp_index.html file.
   b. A navigation list containing an unordered list with the following list items: **Home**, **Menu**, **Events**, and **Catering**. Link the items to the mp_index.html, mp_menu.html, mp_events.html, and mp_catering.html files respectively.
4. Below the `header` element insert an `article` element. Below the `article` element, insert a `footer` element containing the following text:
   Mobile Panini ♨ 31 West Avenue, Charlotte NC 28204 ♨ 704-555-2188
   where ♨ is inserted using the **9832** character code and an extra space is added between NC and **28204** using the `nbsp` character name.
5. Go to the **mp_pages.txt** file in your text editor. This file contains the text content of the four pages in the Mobile Panini website. Copy the text of the Welcome section, which will be used in the home page of the website. Return to **mp_index.html** in your HTML editor and paste the copied text into the `article` element.
6. Within the `article` element, do the following:
   a. Mark the Welcome line as an h1 heading.
   b. Below the `h1` element, insert an inline image containing the mp_photo1.png file with an empty text string for the alternate text.
   c. Mark the next five paragraphs as paragraphs using the p element. Within the first paragraph, mark the text *Mobile Panini* as strong text. Within the third paragraph mark the text *Curbside Thai* as emphasized text.
   d. The fourth paragraph contains Mobile Panini's phone number. Mark the phone number as a telephone link and be sure to include the international code in the URL. Note that this number is fictional, so, if you have access to a mobile browser and want to test the link, you might want to replace this number with your phone number.
   e. The fifth paragraph contains Mobile Panini's email address. Mark the email address as a hypertext link. Once again, note that this email address is fictional, so, if you want to test this link, you will need to replace the Mobile Panini email address with your email address.
7. Save your changes to the file and then open the **mp_index.html** file in your browser. Verify that the layout and appearance of the page resemble that shown in Figure 1–45. If possible, test the telephone links and email links to verify that they open the correct application.
8. Go to the **mp_index.html** file in your HTML editor, and copy the `header` and `footer` elements. Then go to the **mp_menu.html** file in your HTML editor and paste the `header` and `footer` elements into the `body` element so that this page has the same logo and navigation list and footer used in the home page. Insert an `article` element between the header and footer.
9. Return to the **mp_pages.txt** file in your text editor and copy the contents of the Mobile Panini menu. Then, go to the **mp_menu.html** file in your HTML editor and paste the copied text into the `article` element.

10. Within the `article` element of the mp_menu.htm file, do the following:
    a. Mark the text title *Our Menu* as an h1 heading.
    b. Enclose the menu items in a description list with the name of each menu item marked with the `dt` element and each menu description marked with the `dd` element.
11. Save your changes to mp_menu.html file. Open the page in your browser and verify that each menu item name appears in a bold font and is separated from the indented item description by a horizontal line.
12. Go to the **mp_index.html** file in your HTML editor and copy the `header` and `footer` elements. Then, go to the **mp_events.html** file in your HTML editor and paste the `header` and `footer` elements into the `body` element. Insert an `article` element between the header and footer.
13. Return to the **mp_pages.txt** file in your text editor and copy the list of upcoming events at the bottom of the file. Then, go to the **mp_events.html** file in your HTML editor and paste the copied text into the `article` element.
14. Within the `article` element, do the following:
    a. Mark the text *Where Are We This Week*? as an h1 heading.
    b. Enclose each day's worth of events within a separate `div` (or division) element.
    c. Within each of the seven day divisions, enclose the day and date as an h1 heading. Enclose the location within a paragraph element. Insert a line break element, `<br />`, directly before the time of the event so that each time interval is displayed on a new line within the paragraph.
15. Save your changes to mp_events.html file. Open the page in your browser and verify that each calendar event appears in its own box with the day and date rendered as a heading.
16. Go to the **mp_index.html** file in your HTML editor and copy the `header` and `footer` elements. Then, go to the **mp_catering.html** file in your HTML editor and paste the `header` and `footer` elements into the body element. Insert an `article` element between the header and footer and then insert an `aside` element within the article.
17. Directly after the opening `<article>` tag, insert an `h1` element containing the text **Catering**.
18. Return to the **mp_pages.txt** file in your text editor and copy the text about the mobile kitchen, including the heading. Then, go to the **mp_catering.html** file in your HTML editor and paste the copied text into the `aside` element.
19. Within the `article` element, do the following:
    a. Mark the text *About the Mobile Kitchen* as an h1 heading.
    b. Mark the next two paragraphs as paragraphs.
20. Return to the **mp_pages.txt** file in your text editor and copy the text describing Mobile Panini's catering opportunities; do not copy the Catering head. Then, go to the **mp_catering.html** file in your HTML editor and paste the copied text directly after the `aside` element.
21. Make the following edits to the pasted text:
    a. Mark the first two paragraphs as paragraphs.
    b. Enclose the list of the six catering possibilities within an unordered list with each item marked as a list item.
    c. Mark the concluding paragraph as a paragraph.
22. Save your changes to mp_catering.html file. Open the page in your browser and verify that the information about the mobile kitchen appears as a sidebar on the right edge of the article.
23. Return to the **mp_index.html** file in your browser and verify that you can jump from one page to another by clicking the entries in the navigation list at the top of each page.

APPLY

Case Problem 1

Data Files needed for this Case Problem: jtc_index_txt.html, jtc_services_txt.html, 2 CSS files, 3 PNG files, 1 TXT file

Jedds Tree Care Carol Jedds is the owner and operator of Jedds Tree Care and tree removal and landscaping company in Lansing, Michigan. She has asked for your help in developing her company's website. She has already written some of the text for a few sample pages and wants you to write the HTML code. Figure 1–51 shows a preview of the company's home page that you'll create.

Figure 1–51 Jedds Tree Care home page

The style sheets and graphic files have already been created for you. Your job is to write the HTML markup.

Complete the following:

1. Using your editor, open the **jtc_index_txt.html** and **jtc_services_txt.html** files from the html01 ▸ case1 folder. Enter ***your name*** and ***the date*** in the comment section of each file, and save them as **jtc_index.html** and **jtc_services.html** respectively.
2. Go to the **jtc_index.html** file in your HTML editor. Within the document head, do the following:
   a. Use the `meta` element to set the character encoding of the file to **utf-8**.
   b. Set the document title to **Jedds Tree Care**.
   c. Link the document to the jtc_base.css and jtc_layout.css style sheet files.
3. Within the document body, insert a `header` element, an `aside` element, and an `article` element.

4. Within the `header` element, insert a navigation list with links to jtc_index.html and jtc_services.html file. The text of the links should be **home** and **services** respectively.
5. Go to the **jtc_pages.txt** file in your text editor. The first section in the file contains comments made by Jedds Tree Care customers. Copy the text of the three reviews including the reviewer names. Then, go to the **jtc_index.html** file in your HTML editor and paste the copied text within the `aside` element.
6. Within the `aside` element, add the following content and markup:
   a. Directly after the opening `<aside>` tag, insert an inline image for the **jtc_comments.png** file. Specify **Comments** as the alternate text.
   b. Enclose each of the three reviewer comments within a `blockquote` element, including both the text of the quote and the name of the review.
   c. Within each of the three `blockquote` elements,
      i. mark the review as a paragraph.
      ii. mark the line containing the reviewer name as a `cite` element.
      iii. replace the "---" text with the em dash character (—) using the character reference `&mdash;`.
7. Go to the `article` element and insert a `header` element containing the inline image file jtc_photo1.png with the alternate text *Jedds Tree Care*.
8. Return to the **jtc_pages.txt** file in your text editor and copy the second section of text containing the description of the company and its contact information. Then, go to the **jtc_index.html** file in your HTML editor and paste the copied text in the `article` element, directly below the article header.
9. Mark up the content of the page article as follows:
   a. Mark the first two paragraphs using the `<p>` tag.
   b. Enclose the five lines of the contact information within an `address` element. Insert a line break element at the end of the first four lines so that each part of the address appears on a new line in the rendered page.
   c. Mark the text *Jedds Tree Care* in the first line of the address as a `strong` element.
   d. Mark the email address as a hypertext link. Make the telephone number a telephone link, including the international access code.
10. Save your changes to the jtc_index.html file. Open the page in your browser and verify that the layout and contents of the page resemble that shown in Figure 1–51. Note that under the smaller screen widths associated with mobile devices, the text of the reviewer comments is not displayed.
11. Go to the **jtc_services.html** file in your HTML editor. Insert the same metadata in the document head to match what you did for the jtc_index.html file *except* name the page title **Jedds Tree Care Services**.
12. Go to the **jtc_index.html** file in your HTML editor and copy the body header. Then, go to the **jtc_services.html** file and paste the copied header into the document body so that both files share a common header design.
13. Return to the **jtc_pages.txt** file in your text editor and copy the content of the third section, which contains information on the services offered by Jedds Tree Care. Be sure to copy the heading as well. Then, go to the **jtc_services.html** file in your HTML editor and paste the copied text directly after the header.
14. Mark the content describing Jedds Tree Care services as follows:
   a. Mark the heading *Jedds Tree Care Services* as an h1 heading.
   b. Directly after the h1 element, insert an inline image file for the **jtc_photo2.png** with the alternate text set to an empty text string ("").
   c. Mark each of the headings associated with individual services as h2 headings.
   d. Mark each service description as a paragraph.

15. Directly after the text of the last service, insert a `footer` element containing the following text:
Jedds Tree Care ◆ 201 Edward Ave. ◆ Lansing, MI 48930
where the ◆ symbol is inserted using the character reference ♦.
16. Save your changes to the file and open the **jtc_services.html** file in your browser. Verify that the page title is displayed as a major heading and the name of each service is displayed as a second level heading.

CHALLENGE

Case Problem 2

Data Files needed for this Case Problem: dr_index_txt.html, dr_info_txt.htm, dr_faq_txt.html, 4 CSS files, 2 PNG files, 3 TXT files

Diane's Run *Diane's Run* is a charity run to raise money for breast cancer awareness and research funding. Peter Wheaton is the charity run's organizer and he has asked you to help modify the run's website. He has revised text that he wants added to the current site. A preview of the page you'll create is shown in Figure 1–52.

Figure 1–52 Diane's Run home page

Peter has supplied you with the text content, the graphic images, and style sheets you need for the project. Your job will be to write HTML code for three pages: the site's home page, a page containing race information, and finally a page containing a list of frequently asked questions (FAQ).

Complete the following:

1. Using your editor, open the **dr_index_txt.html**, **dr_info_txt.html**, and **dr_faq_txt.html** files from the html01 ▸ case2 folder. Enter ***your name*** and ***the date*** in the comment section of each file, and save them as **dr_index.html**, **dr_info.html**, and **dr_faq.html** respectively.
2. Go to the **dr_index.html** file in your HTML editor. Within the document head, add the following metadata:
   a. Set the character encoding of the file to **utf-8**.
   b. Insert the search keywords: **breast cancer**, **run**, **race**, and **charity**.
   c. Set the title of the document to **Diane's Run**.
   d. Link the document to the dr_base.css and dr_layout.css style sheet files.
3. Within the document body, insert a `header` element, two `section` elements, and a `footer` element.
4. In the `header` element, insert a navigation list containing an unordered list with the items: **Home**, **Race Info**, and **FAQ**. Link the items to the dr_index.html, dr_info.html, and dr_faq.html files respectively.
5. The file dr_index.txt contains the text to be inserted into the Diane's Run home page. Go to the **dr_index.txt** file in your text editor and copy the text from the first section of the file. Then, go to the **dr_index.html** file in your HTML editor and paste it into the first `section` element.
6. Add the following markup to the content of the first `section` element:
   a. Mark the line *What Your Support Does* as an h1 heading.
   b. Mark the next two paragraphs as paragraphs using the `p` element.
   c. Mark the four ways a contribution can help as an unordered list. Mark the dollar amounts of each list item using the `strong` element.
7. Return to the **dr_index.txt** file in your text editor, copy the text from the second section, then close the dr_index.txt file. Go to the **dr_index.html** file in your HTML editor and paste the copied text within the second `section` element.
8. Within the second `section` element in the dr_index.html file, add the following:
   a. Enclose the opening heading *Diane's Run - September 12, 2021* within a `header` element and marked as an h1 heading. Directly above this heading, insert the inline image file **dr_photo1.png** with **Diane's Run** as the alternate text of the image.
   b. Mark the first paragraph after the header as a paragraph. Mark the text *Diane's Run* in this opening paragraph using the `strong` element.
   c. Mark the minor headings *How to Join*, *History*, and *Remembering Diane* as h2 headings. Mark the other blocks of text as paragraphs.
9. Within the `footer` element, insert the following text:
   Diane's Run ♥ 45 Mountain Drive ♥ Cheyenne, WY 82001
   where the ♥ character is inserted using the character reference ♥.
10. Save your changes to the file and then open **dr_index.html** in your browser. Verify that the content and the layout of the page resemble that shown in Figure 1–52.
11. Go to the **dr_info.html** file in your HTML editor. Within the document head, link the page to the dr_base.css and dr_layout2.css style sheets.
12. Go to the **dr_index.html** file in your HTML editor and copy the body header content. Then, go to the **dr_info.html** file in your HTML editor and paste the copied content into the document body. Repeat for the body footer so that the Racing Information page has the same navigation list and footer as the home page. Between the `header` and `footer` element, insert a `section` element.

Explore 13. Within the `section` element, insert a `header` element with the following content:
   a. Insert a paragraph with the text **Page last updated: Tuesday, August 31, 2021.** Mark the date using the `time` element with the `datetime` attribute equal to **2021-08-31**.
   b. Add the text **Race Information** as an h1 heading.
   c. Insert the inline image file dr_logo.png with **Diane's Run** as the alternate text.
14. Go to the **dr_info.txt** file in your text editor. This file contains the text describing the race. Copy the content describing the race from the file, then close the dr_info.txt file. Go to the dr_info.html file in your HTML editor and paste the copied text into the `section` element, directly after the section header.
15. Mark the content of the `section` element as follows:
   a. Mark the opening block of text directly after the section header as a paragraph.
   b. Mark the headings *Race Times*, *Goodies and Stuff*, and *Notes* as h2 headings.
   c. Below each of the `h2` elements, mark the list of items that follows as an unordered list.
16. Save your changes to the file and then load **dr_info.html** in your browser to verify that the layout and content are readable.
17. Go to the **dr_faq.html** file in your HTML editor. Within the document head, link the page to the dr_base.css and dr_layout3.css style sheets.
18. Go to the **dr_index.html** file in your HTML editor and copy the body header content. Then, go to the **dr_faq.html** file in your HTML editor and paste the copied content into the document body. Repeat with the body footer so that the FAQ page has the same navigation list and footer as was used in the home page. Between the `header` and `footer` element, insert a `section` element.
19. Within the `section` element, insert a `header` element with the `id` attribute **pagetop**. Within the header, insert the inline image file dr_logo.png with the alternate text **Diane's Run** followed by the `h1` element with the text **Frequently Asked Questions**.
20. Go to the **dr_faq.txt** file in your text editor. This file contains a list of frequently asked questions followed by the question answers. Copy the text and then close the dr_faq.txt file. Then, go to the **dr_faq.html** file in your HTML editor and paste the copied text into the `section` element, directly after the section header.

Explore 21. Next, you'll create a series of hypertext links between the list of questions and their answers within the same document. Make the following changes to the `section` element in the dr_faq.html file:
   a. Mark the 13 questions at the top of the section as an ordered list.
   b. Notice that below the ordered list you just created, the questions are repeated and each question is followed by its answer. Mark the text of those questions as an h2 heading and the answer as a paragraph. Add an `id` attribute to each of the 13 h2 headings with the first heading given the id **faq1**, the second heading **faq2**, and so forth down to **faq13** for the last h2 heading.
   c. After the last answer, insert a paragraph with the text **Return to the Top** and mark the text as a hypertext link pointing to the `header` element with the id **pagetop**.
   d. Return to the ordered list at the top of the section that you created in Step a. Change each item in the ordered list to a hypertext link pointing to the h2 heading containing the question's answer that you created in Step b. For example, the first question *How do I sign up?* should be linked to the h2 heading with the faq1 id.
22. Save your changes to the file and then open **dr_faq.html** in your browser. Verify that by clicking a question within the ordered list, the browser jumps to that question's answer. Further, verify that clicking the Return to the Top link at the bottom of the page causes the browser to return to the top of the page.

Explore 23. Return to the **dr_index.html** file in your HTML editor. Add the following two hypertext links to the *How to Join* paragraph in the second `section` element:

a. Change the email address *dianesrun@example.com* to an email link with the subject heading Entry Form.

b. Change the word *accommodations* to a hypertext link pointing to the element with the id faq13 in the dr_faq.html file.

24. Save your changes to the file and reload dr_index.html in your browser. Verify that clicking the email link brings up your email program with the email address and the subject heading already filled in.

25. Click the accommodations hypertext link and verify that the browser goes to the last answer on the FAQ page.

26. Verify that you can jump between all three pages by clicking the navigation links at the top of the page.