

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

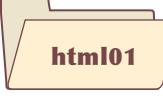
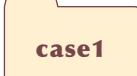
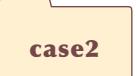
Getting Started with HTML 5

Creating a Website for a Food Vendor

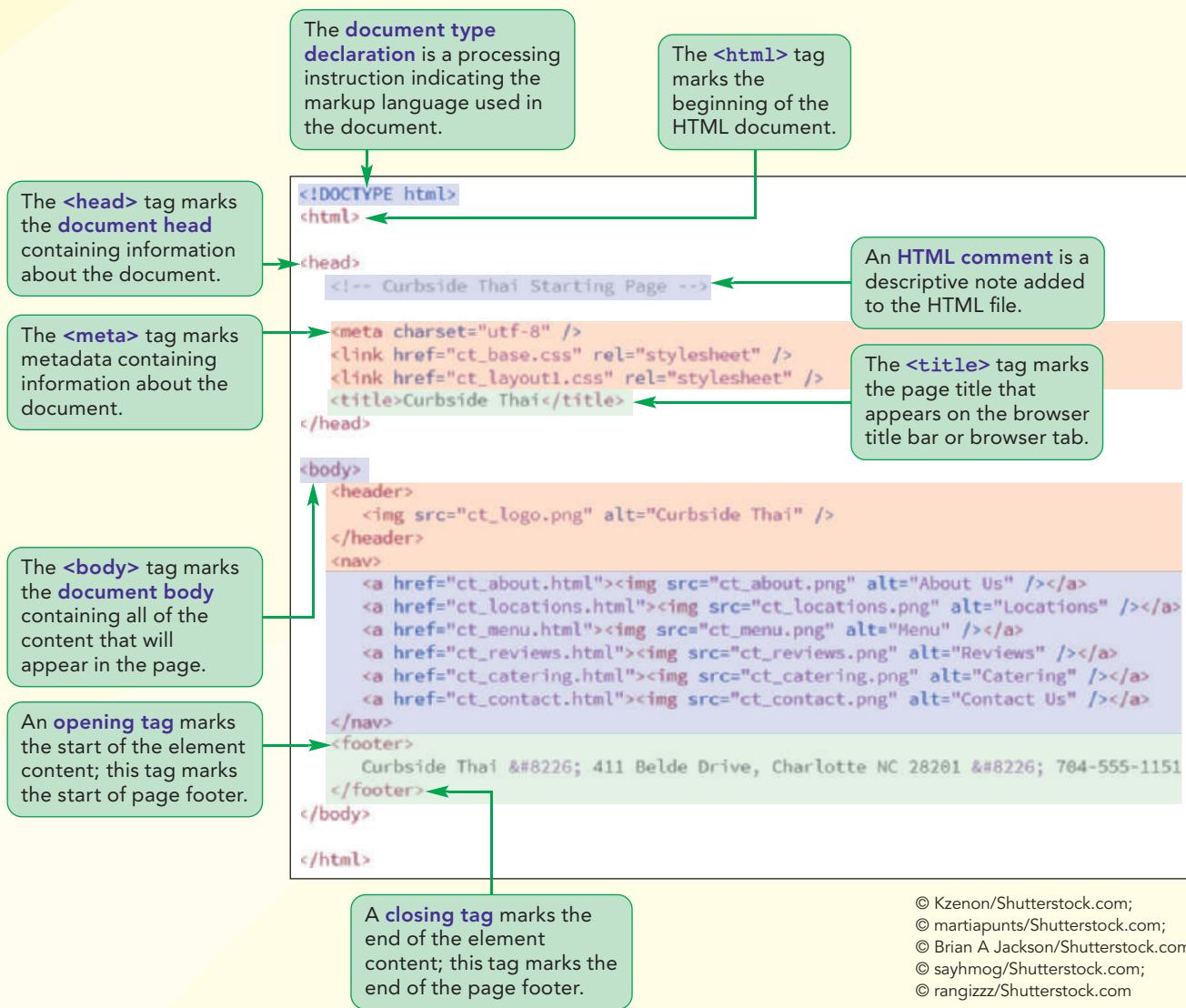
Case | *Curbside Thai*

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	review	code1
	ct_catering_txt.html ct_contact_txt.html ct_locations_txt.html ct_menu_txt.html ct_reviews_.txt.html + 16 files	mp_catering_txt.html mp_events_txt.html mp_index_txt.html mp_menu_txt.html + 5 files	code1-1_txt.html
	code2		
	code1-2_txt.html	code1-3_txt.html + 7 files	code1-4_txt.html + 2 files
	case1		
	jtc_index_txt.html jtc_services_txt.html + 6 files	dr_faq_txt.html dr_index_txt.html dr_info_txt.html + 9 files	demo_characters.html demo_html.html + 3 files

Session 1.1 Visual Overview:



© Kzenon/Shutterstock.com;
 © martiapunts/Shutterstock.com;
 © Brian A Jackson/Shutterstock.com;
 © sayhmog/Shutterstock.com;
 © rangizzz/Shutterstock.com

The Structure of an HTML Document



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

```

<!DOCTYPE html>
<html>
  <head>
    <title>Curbside Thai</title>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <link href="ct_base.css" rel="stylesheet" />
    <link href="ct_layout1.css" rel="stylesheet" />
  </head>
  <body>
    <header>
      
    </header>
    <nav>
      <a href="ct_about.html"></a>
      <a href="ct_locations.html"></a>
      <a href="ct_menu.html"></a>
      <a href="ct_reviews.html"></a>
      <a href="ct_catering.html"></a>
      <a href="ct_contact.html"></a>
    </nav>
    <footer>
      Curbside Thai &#8226; 411 Beld Drive, Charlotte NC 28201 &#8226; 704-555-1151
    </footer>
  </body>
</html>

```

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>
```

Copyright 2021 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s). Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

<!DOCTYPE html>

<!doctype html>

both are okay

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

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 Curbside Thai.</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

© Kzenon/Shutterstock.com; © martipapunits/Shutterstock.com;
 © Brian A. Jackson/Shutterstock.com; © sayhmog/Shutterstock.com;
 © rangizzza/Shutterstock.com; BenBois/openclipart;
 Jmlevick/openclipart

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.

TIP

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

To begin writing the HTML file:

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:

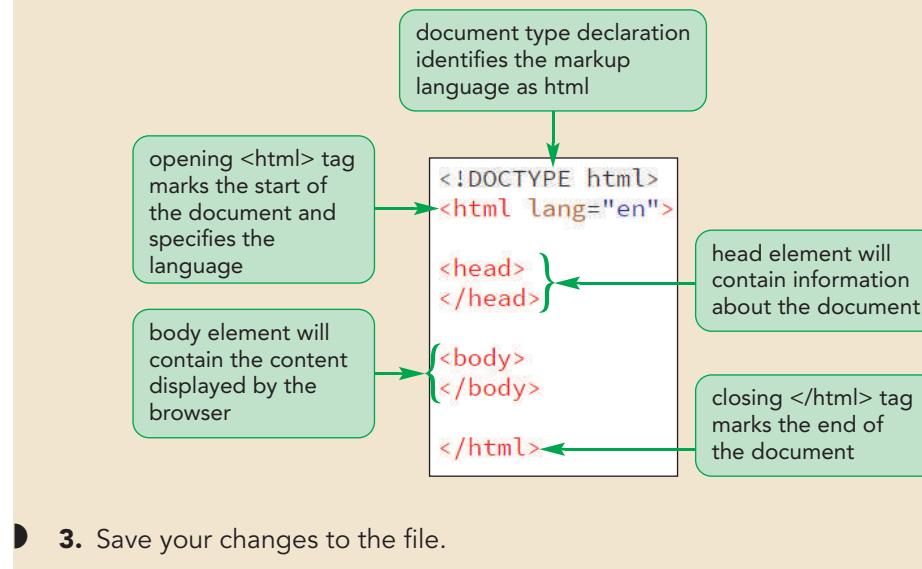
```
<!DOCTYPE html>
<html lang="en">

<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**

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 <u>external resource</u> 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.

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.

To insert the document title:

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

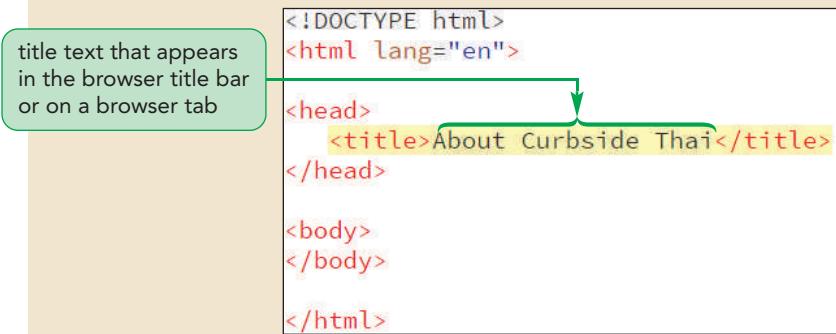
```
<title>About Curbside Thai</title>
```

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

Figure 1–7

Entering the document title

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>About Curbside Thai</title>
  </head>
  <body>
  </body>
</html>
```



- 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
<code>charset="encoding"</code>	Specifies the character encoding used in the HTML document
<code>content="text"</code>	Provides the value associated with the <code>http-equiv</code> or <code>name</code> attributes
<code>http-equiv="content-type default-style refresh"</code>	Provides an HTTP header for the document's content, default style, or refresh interval (in seconds)
<code>name="text"</code>	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`.

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

character encoding used in the document

keywords used for search engines

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

- 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.

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.

REFERENCE

<head>

<!-- 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:

```
<!--
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

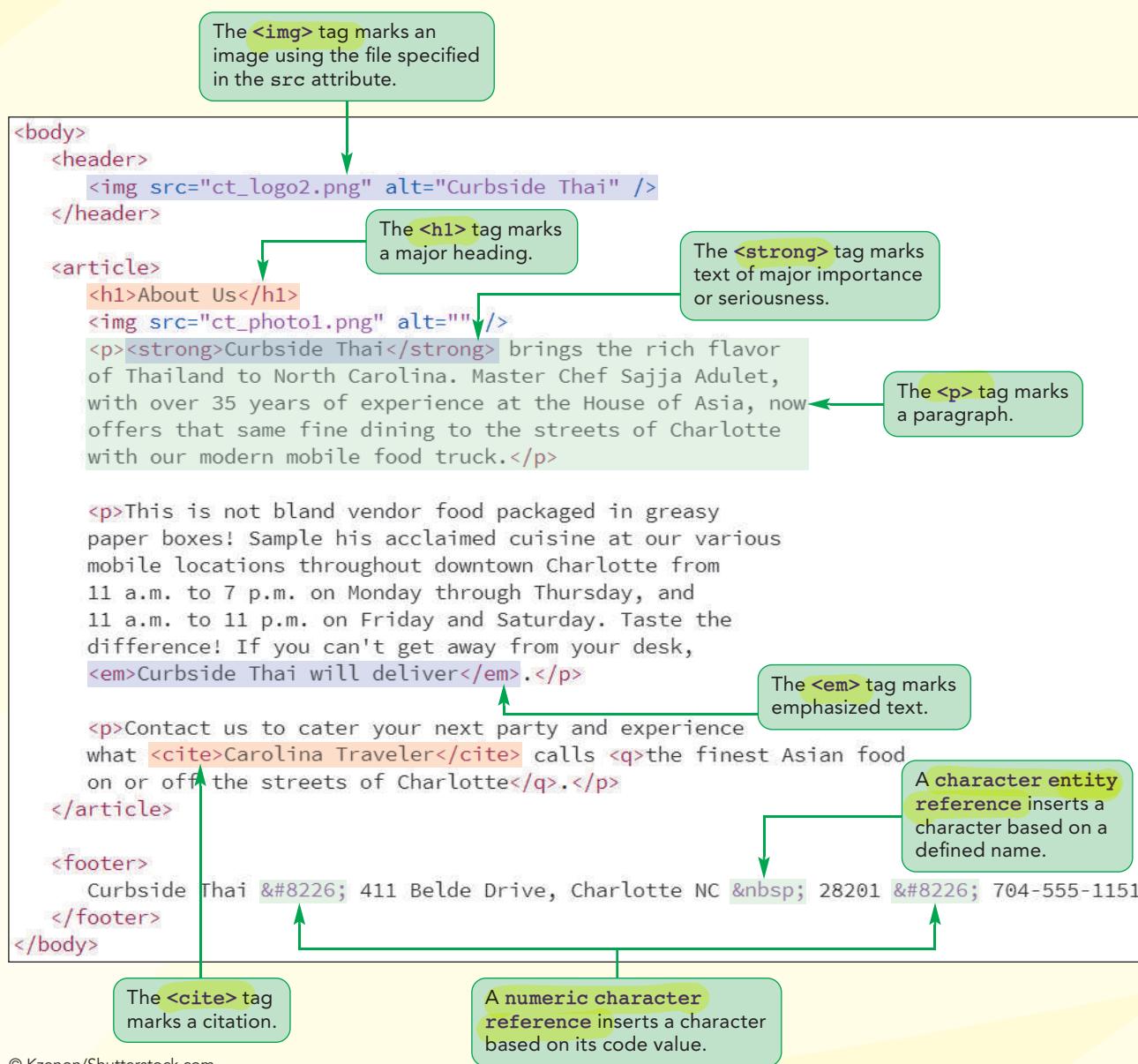
1. What is a markup language?
 - a. A language used for e-commerce websites
 - b.** A language describing the content and structure of a document by tagging document elements
 - c. A language introduced by Microsoft for use in web browsers
 - d. A language that defines the appearance of web pages for 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 web page</p>
 - b.** <p>Welcome to my web page</p>
 - c. <p>
 Welcome to my
 web page
 </p>
 - d. <p>Welcome to my
 web page
 </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.

T

Session 1.2 Visual Overview:



© Kzenon/Shutterstock.com

HTML Page Elements

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

Images are added to the web page.

About Us

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

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

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.*

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.

Nonbreaking space is inserted with the ` ` character entity reference.

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.

Carolina Traveler



Kzenon/Shutterstock.com

footer

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

similar
to <p>
<h> ...

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.

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:

```

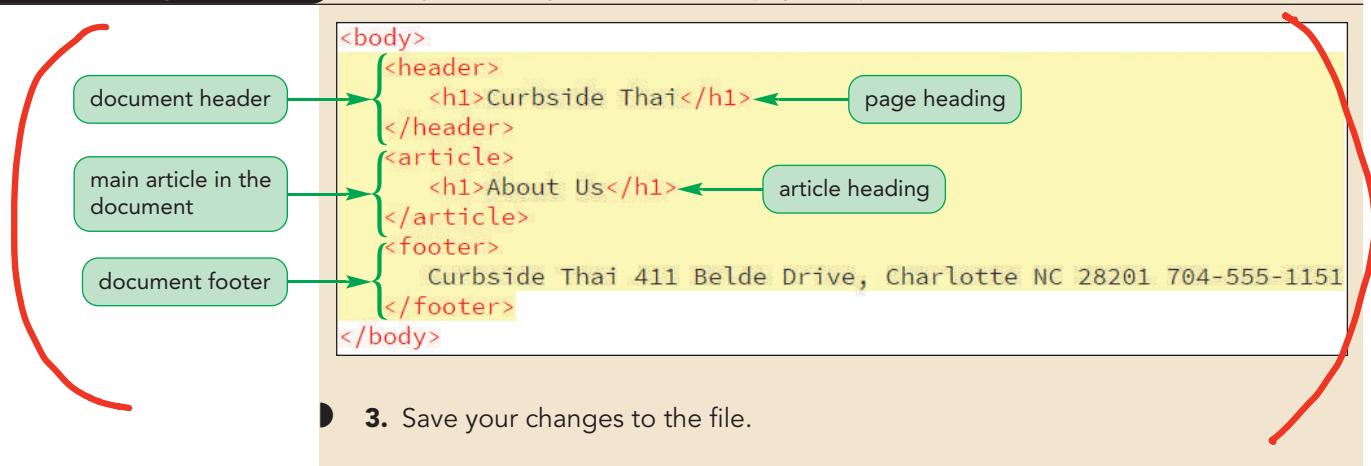
<header>
  <h1>Curbside Thai</h1>
</header>
<article>
  <h1>About Us</h1>
</article>
<footer>
  Curbside Thai 411 Belde Drive, Charlotte NC 28201 704-555-1151
</footer>

```

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

Figure 1-12

Adding sectioning elements to the page body



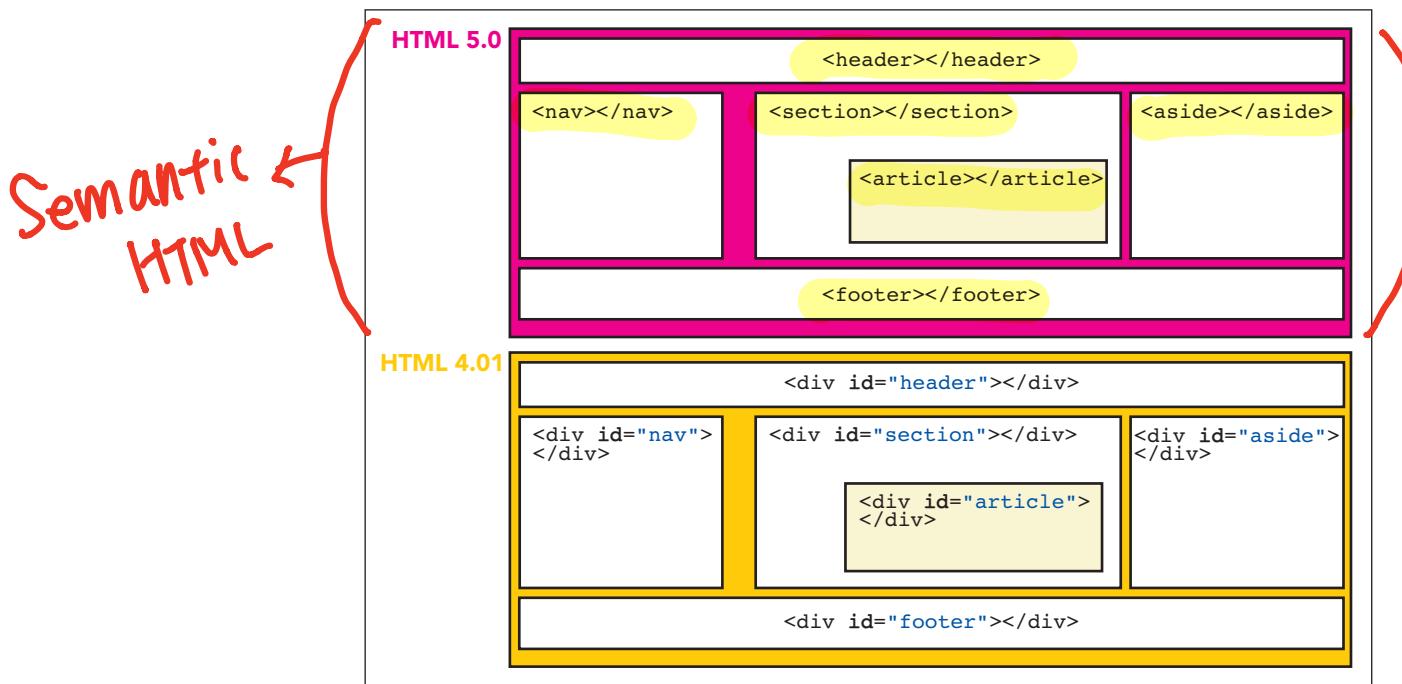
Comparing Sections in HTML 4 and HTML 5

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

don't need
to memorize
all 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 `<p>` tag and a closing `</p>` 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

```

<article>
  <h1>About Us</h1>
  <p>Curbside Thai 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, Curbside Thai will deliver.</p>
  <p>Contact us to cater your next party and experience what Carolina Traveler calls the finest Asian food on or off the streets of Charlotte.</p>
</article>

```

first paragraph

second paragraph

third paragraph

each paragraph is enclosed within an opening `<p>` tag and a closing `</p>` tag

- 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>
```

4x
link is
meta data

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 `` 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

```

<article>
  <h1>About Us</h1>
  <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 <code><cite>Carolina Traveler</cite></code> calls <q>the finest Asian food
    on or off the streets of Charlotte</q>.</p>
</article>

```

strong and important text marked with the `` tag

emphasized text marked with the `` tag

citation marked with the `<code><cite>` tag

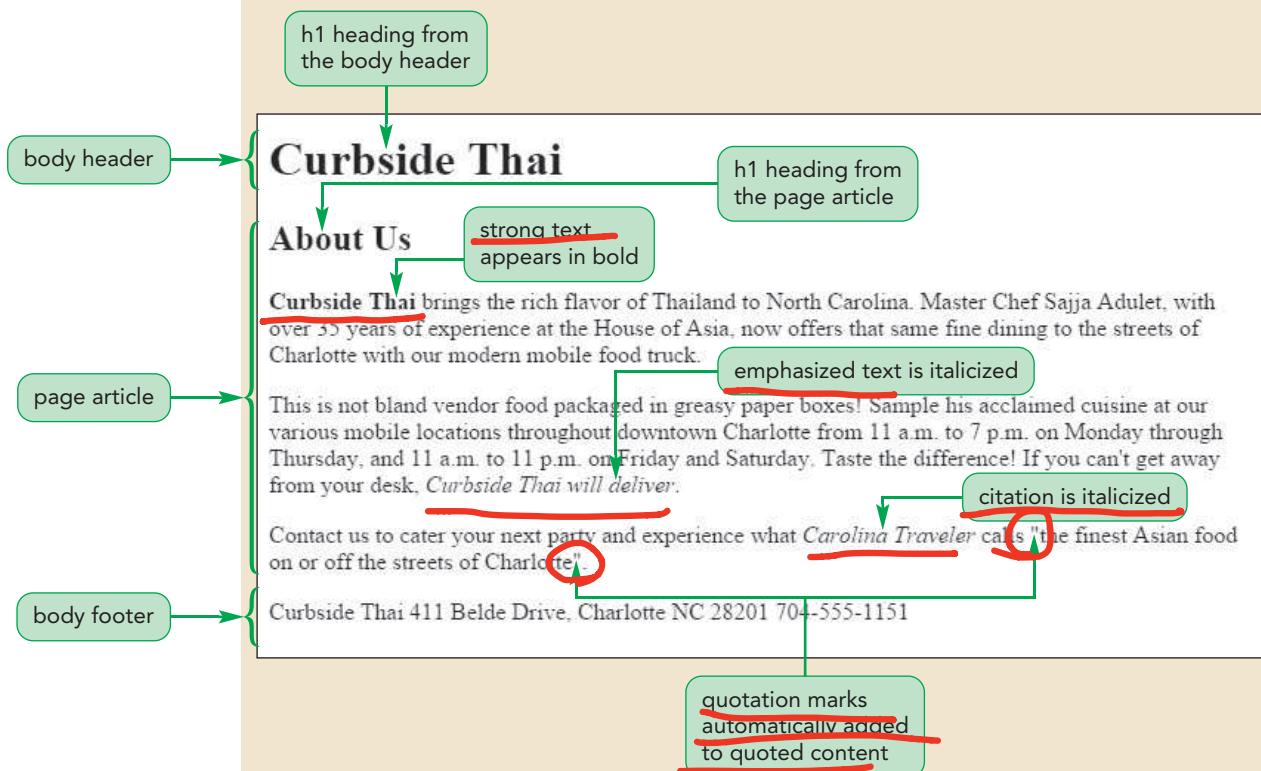
quoted material marked with the `<q>` tag

- 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**.

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.

TRY IT
You can explore the impact of different HTML tags using the `demo_html.html` file in the `html01 ▶ demo` folder.

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:

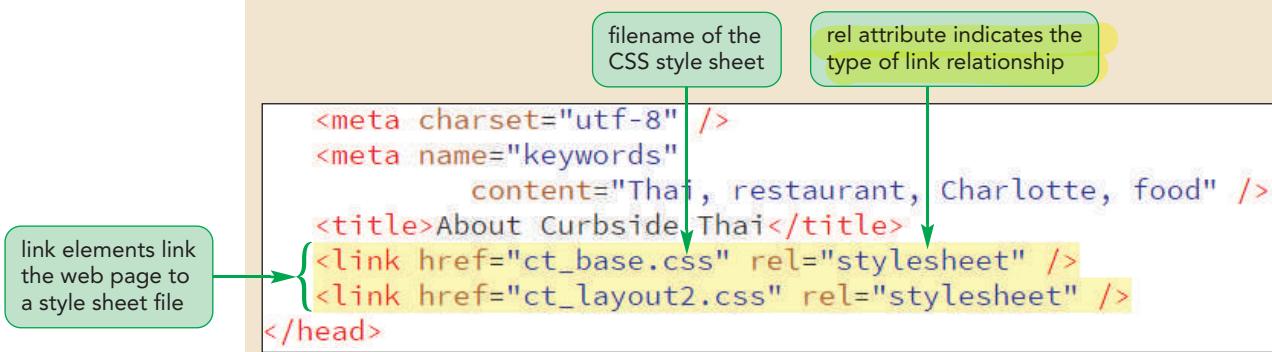
- 1. Return to the `ct_about.html` file in your HTML editor.
- 2. Directly before the closing `</head>` tag, insert the following `link` elements:

```
<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

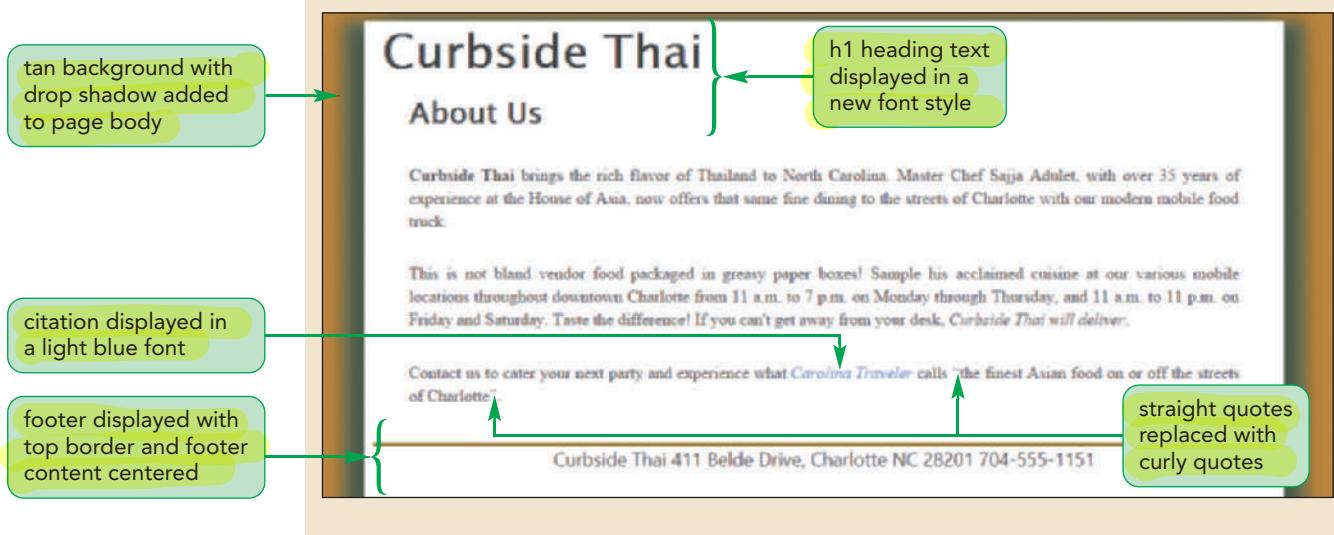
Linking to style sheets



- 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

©

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:

©

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.

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

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 **•** directly after the word *Thai* and after the postal code **28201**. Insert the character reference **&nbsp**, directly before the postal code.

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

Inserting special characters

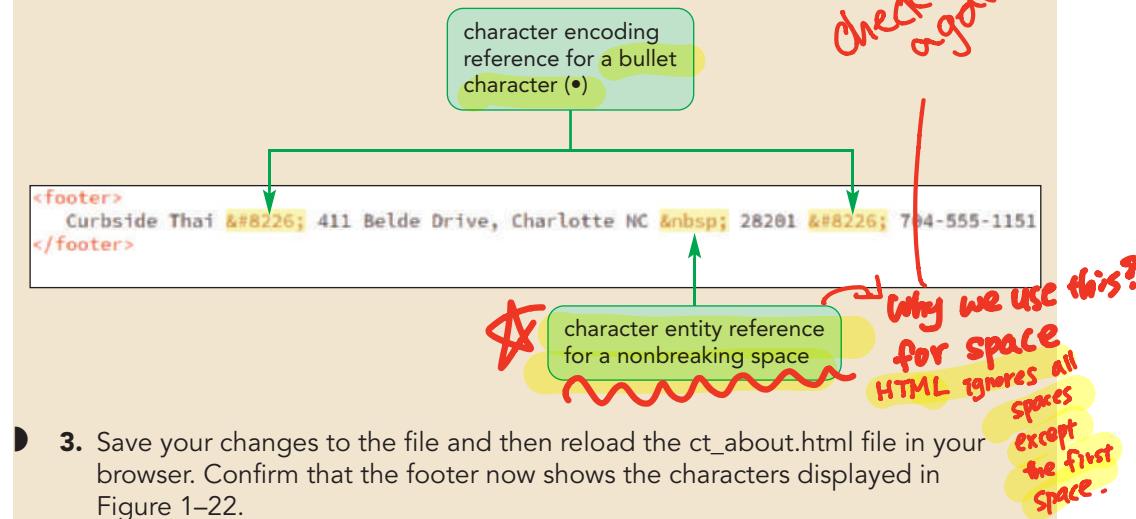
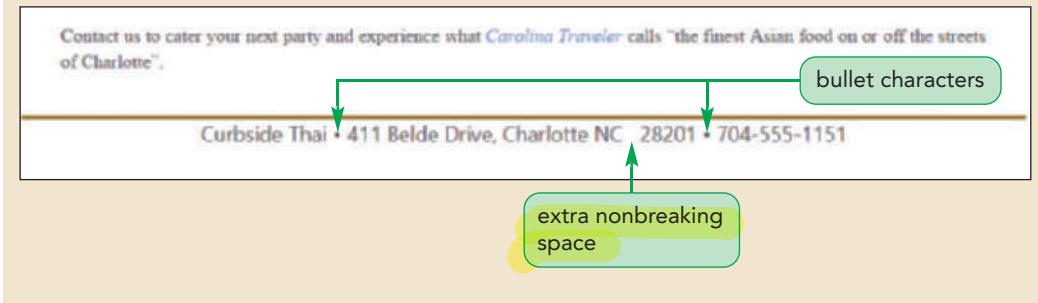


Figure 1–22

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

Self-closing
``

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

```

```

Q. If we define height and width in both html and css, who will dominate?

→ Generally Speaking, html inline dominates.

``

REFERENCE

Images marked with the `` 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"`

→ manipulate image size

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.

Embedding an Inline Image

- To embed an inline image into the document, use

``

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

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

Figure 1–24

Inserting inline images

image added to the About Us article

h1 heading replaced with an inline image

```
<header>
  
</header>
<article>
  <h1>About Us</h1>
  
  <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>

```

- 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

About Us

Curbside Thai 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.

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, *Curbside Thai will deliver.*

© Kzenon/Shutterstock.com

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

`
`

Line breaks are placed within grouping elements, such as 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

```

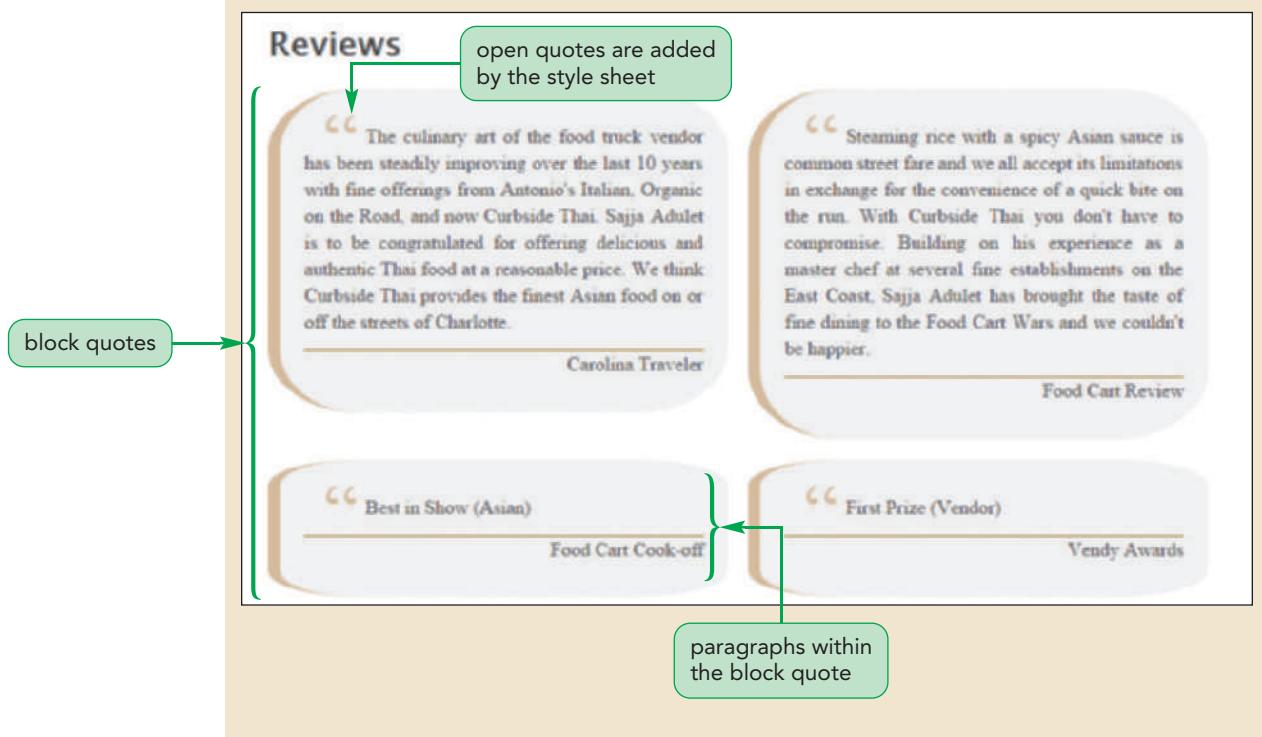
<article>
  <h1>Reviews</h1>
  <blockquote>
    <p>The culinary art of the food truck vendor has been steadily improving over the last 10 years with fine offerings from Antonio's Italian, Organic on the Road, and now Curbside Thai. Sajja Adulet is to be congratulated for offering delicious and authentic 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>

```

- 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>
  
  <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

paragraphs

inline image

Catering

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.

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.



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.

Impress your friends and co-workers with a Curbside Thai-catered event!

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

```

<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>

```

address element to mark up a mailing address

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

- 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

Contact Us

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!

Curbside Thai
411 Belde Drive
Charlotte NC 28201

Call: (704) 555-1151

Email: curbside.thai@example.com

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.

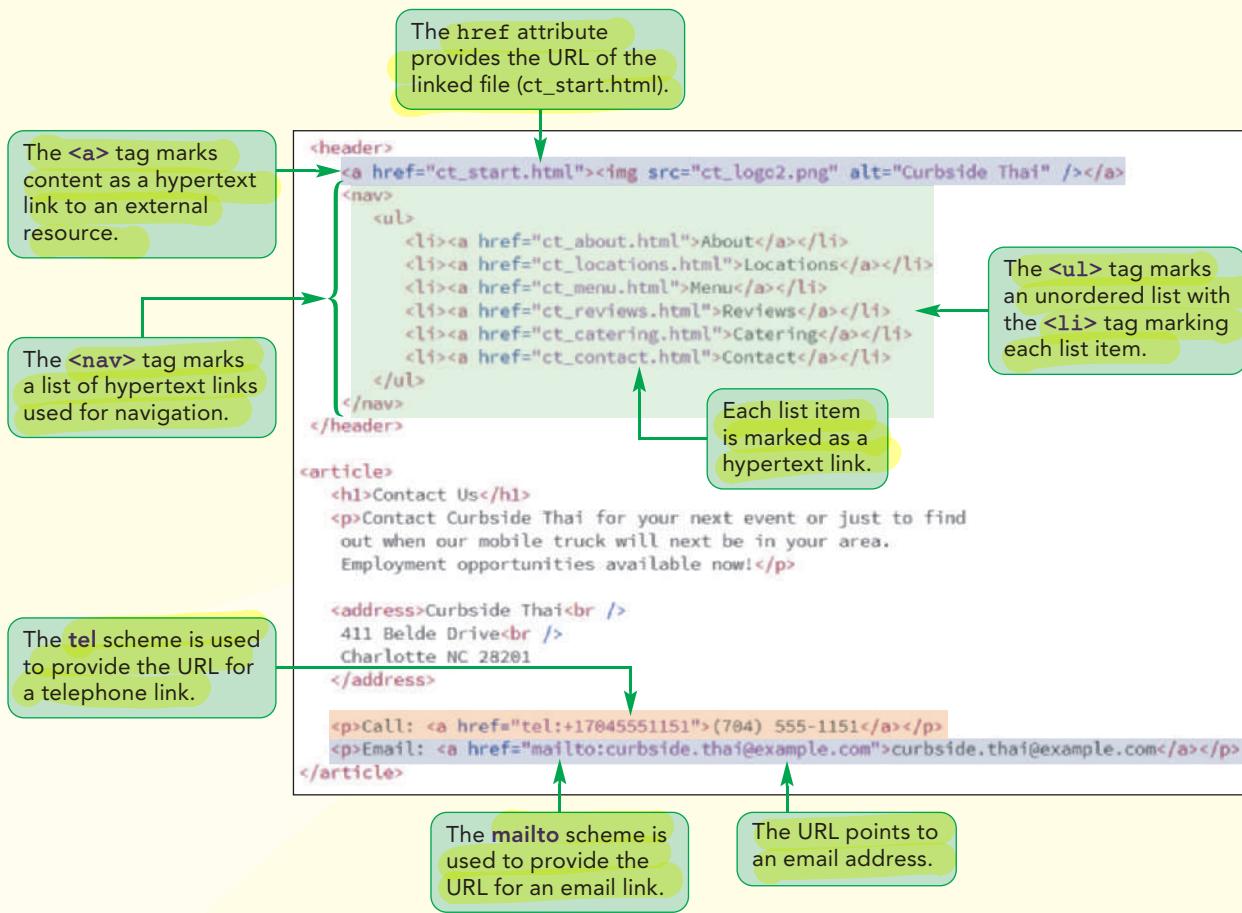
ARIA is a useful tool for enhancing the accessibility of your web page and making the rich resource that is the World Wide Web open to all. A side benefit is that accessibility and usability go hand-in-hand. A website that is highly accessible is also highly usable and that is of value to all users.

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.

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. `Daily Special`**
 - c. `Daily Special`
 - d. `Daily Special`
5. Which of the following should be used to mark the text string "H₂SO₄"?
 - a. `H<lower>2</lower>SO<lower>4</lower>`
 - b. `H²SO⁴`
 - c. `H<code>2</code>SO<code>4</code>`
 - d. `H₂SO₄`**
6. Which of the following links the HTML file to the CSS style sheet file, `mystyles.css`?
 - a. `<stylesheet rel="link">mystyles.css</stylesheet>`
 - 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. `
`
 - c. `<aside />`**
 - d. ``
8. Which character reference is used to insert a nonbreaking space within a text string?
 - a. `&space;`
 - b. ` `**
 - c. `&ws;`
 - d. `©`
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. `Art World`
 - c. `<figure src="awlogo.png" alt="Art World" />`
 - d. ``**

Session 1.3 Visual Overview:



Lists and Hypertext Links

The screenshot shows the 'Contact Us' page of the Curbside Thai website. The page features a navigation bar with links for About, Locations, Menu, Reviews, Catering, and Contact. The main content area includes the restaurant's address (411 Belde Drive, Charlotte NC 28201), a telephone link ('Call: (704) 555-1151'), an email link ('Email: curbside.thai@example.com'), and a footer with the address and phone number. Callout boxes provide the following information:

- Clicking the logo jumps the user to the ct_start.html file.
- The navigation list encloses links to pages in the Curbside Thai website.
- The telephone link opens a telephony application when clicked.
- The email link opens an email program when clicked.

A red annotation on the right side of the page reads: "This page is made by CSS".

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

the name of each menu item is marked as a description term; information about the item is marked as a description

description list

```
<article>
  <h1>Mobile Menu</h1>
  <dl>
    <dt>Basil Beef Sesame Salad</dt>
    <dd>Spicy Angus beef and sweet basil on top of fresh spring mix, red cabbage, carrot, cucumber and tomatoes; served with sesame vinaigrette ($9.95)</dd>
    <dt>Curbside Rice</dt>
    <dd>Stir-fried rice with onions, red bell peppers, peas and carrots, garnished with red cabbage, cucumbers, scallions, and fried garlic ($6.50); add chicken ($8.50) or shrimp ($9.50)</dd>
    <dt>Garlic Pepper Pork</dt>
    <dd>Marinated pork stir-fried with fresh garlic and pepper; served with steamed Jasmine rice, red cabbage, carrot, cucumbers, scallions, and fried garlic ($8.50)</dd>
    <dt>Pad Thai</dt>
    <dd>Stir-fried rice noodles with bean sprouts and chives, garnished with red cabbage, carrot, scallions, lime, and crushed peanuts ($7.50); add chicken ($8.50) or shrimp ($9.50)</dd>
    <dt>Thai Red Curry</dt>
    <dd>Traditional red curry sauce cooked in coconut milk with bamboo shoots, fresh basil, lime, and Thai chili and served on a bed of steamed Jasmine rice ($7.50); add chicken ($8.50) or shrimp ($9.50)</dd>
  </dl>
</article>
```

- 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:

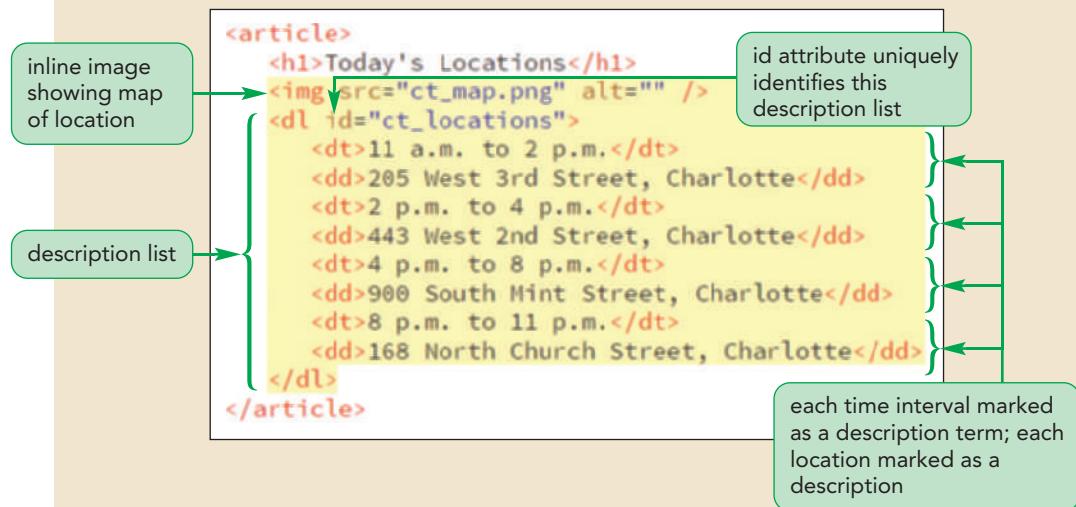
- 1. Open the **ct_locations_txt.html** file from the **html01 ▶ tutorial** folder in your HTML editor. Enter **your name** and **the date** in the comment section and save the file as **ct_locations.html**.
 - 2. Return to the **ct_pages.txt** file in your text editor and copy the four locations from 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:
- ```

```

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

Figure 1–34

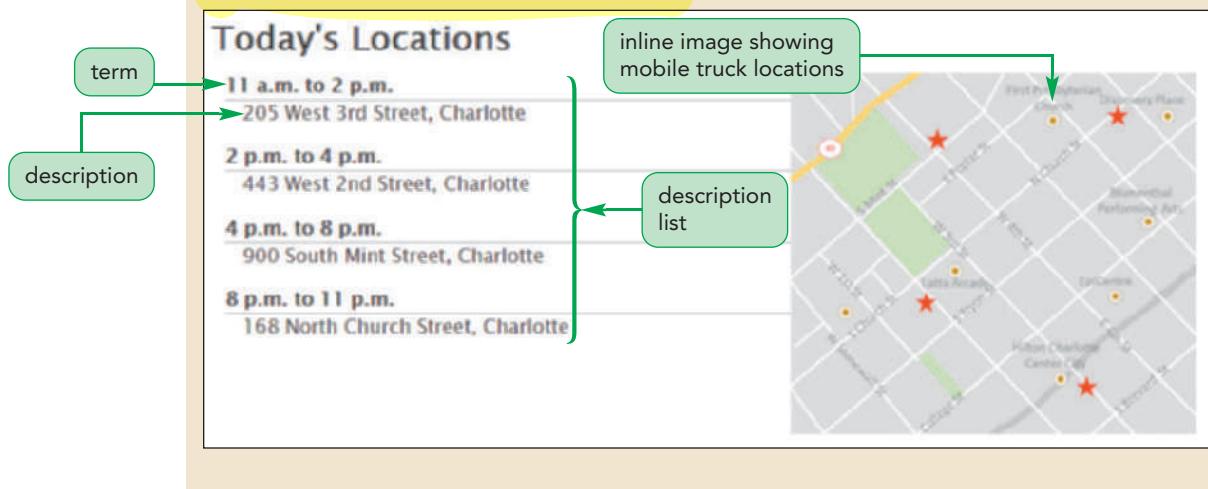
Creating a description list



- 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>

 link1
 link2
 ...

</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>

 About
 Locations
 Menu
 Reviews
 Catering
 Contact

</nav>
```

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

Figure 1–36

### Creating a navigation list

navigation list section created with the nav element

```
<body>
 <header>

 <nav>

 About
 Locations
 Menu
 Reviews
 Catering
 Contact

 </nav>
 </header>
```

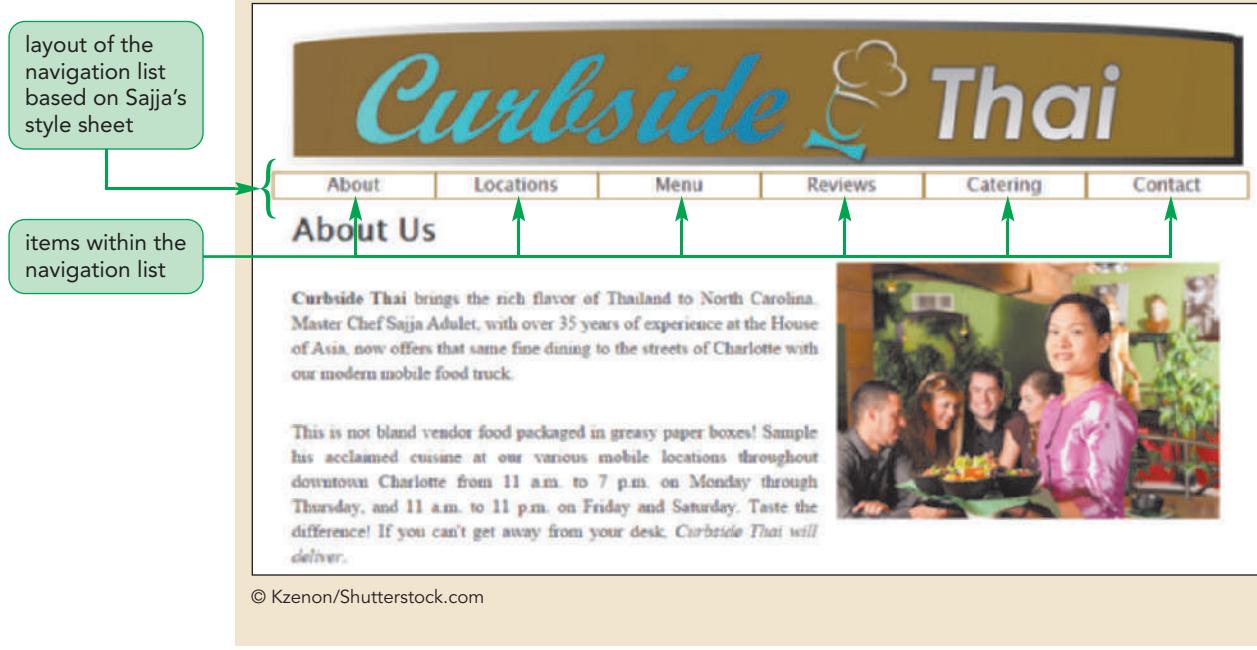
unordered list within the nav section

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

### TIP

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

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

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.

### Marking a Hypertext Link

- To mark content as a hypertext link, use

```
content
```

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
 

```
About
```
- 3. Change the code of the second list item to
 

```
Locations
```
- 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

each item in the navigation list is marked as a hypertext link

```

<nav>

 About
 Locations
 Menu
 Reviews
 Catering
 Contact

</nav>

```

linked file or website address that you want to link

linked file

hypertext

- 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

```

```

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>

 <nav>

```

- 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

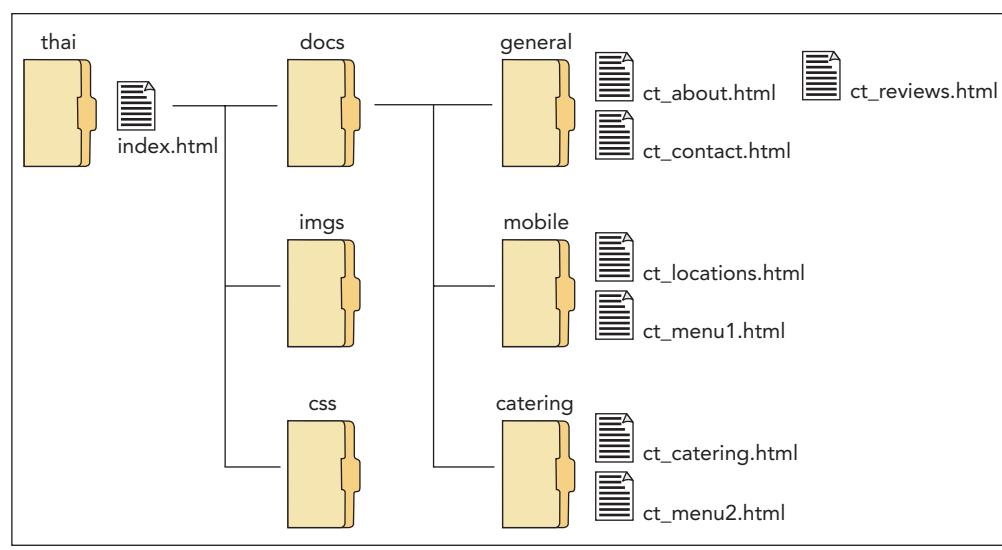
You can give your websites a uniform design by including the same navigation list on each page so that users can easily move from one page to the next.

## 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



in different folders

## 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.

in the same folder

## 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:

```
content
```

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.

```
View our Lunch Menu
```

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:

```
View our Lunch Menu
```

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

```
content
```

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>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.

### 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

```
content
```

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

```
content
```

## 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](http://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:

```

 curbside.thai@example.com

```

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

mailto scheme indicates that this is an email link

email address

email address marked as a hyperlink

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

- 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:

```

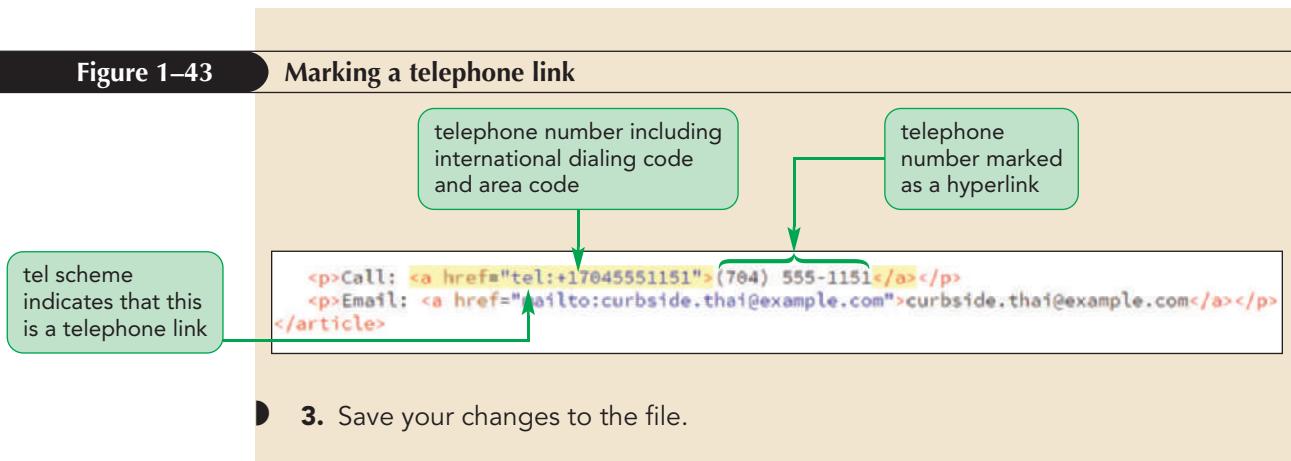
 (704) 555-1151

```

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



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
<code>href="url"</code>	Provides the <code>url</code> of the hypertext link
<code>target=_blank   _parent   _self   _top</code>	Specifies where to open the linked document
<code>download="filename"</code>	Indicates that the link should be downloaded as a file, where <code>filename</code> is the name given to the downloaded file
<code>rel="type"</code>	Provides the relationship between the linked document and the current page
<code>hreflang="lang"</code>	Indicates the language of the linked document
<code>type="mime-type"</code>	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:

```
content
```

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.

→ **validator.w3.org**

### **Validating Your Website**

( Go to this website  
and put html files  
and check )

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 ``.
- 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.

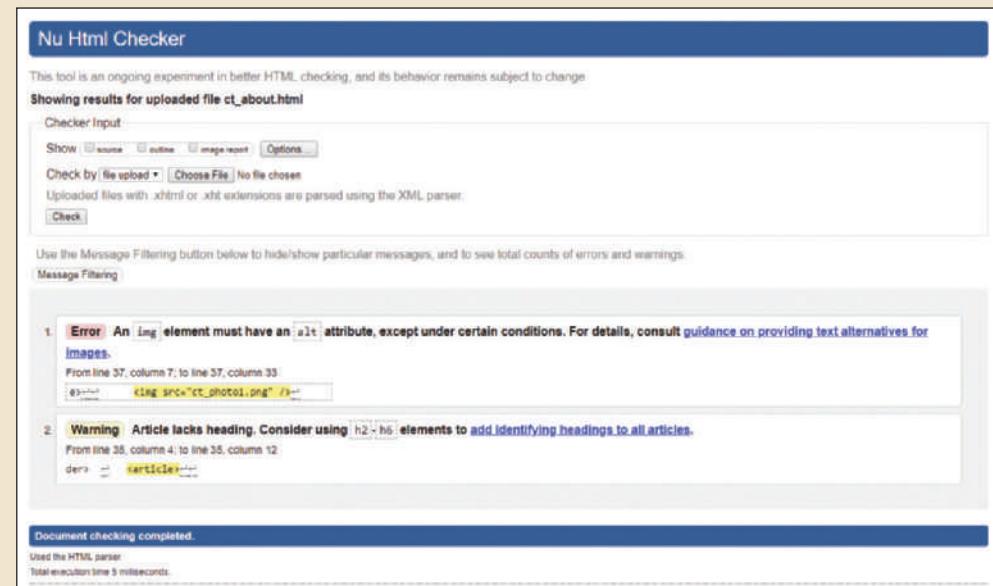
### To validate the code in the **ct\_about.html** file:

- 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_photo1.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 following 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://www.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`

**OBJECTIVES****Session 2.1**

- Explore the history of CSS
- Study different types of style sheets
- Explore style precedence and inheritance
- Apply colors in CSS

**Session 2.2**

- Use contextual selectors
- Work with attribute selectors
- Apply text and font styles
- Use a web font

**Session 2.3**

- Define list styles
- Work with margins and padding space
- Use pseudo-classes and pseudo-elements
- Insert page content with CSS

# Getting Started with CSS

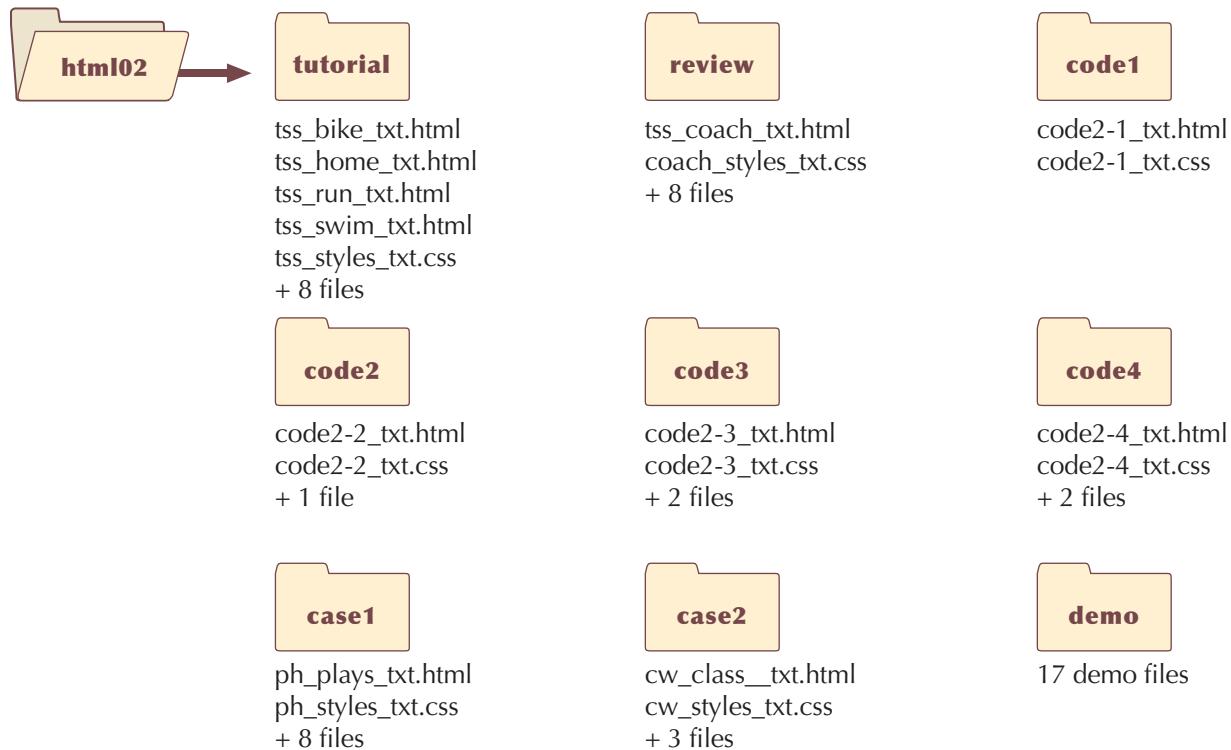
## *Designing a Website for a Fitness Club*

### **Case | *Tri and Succeed Sports***

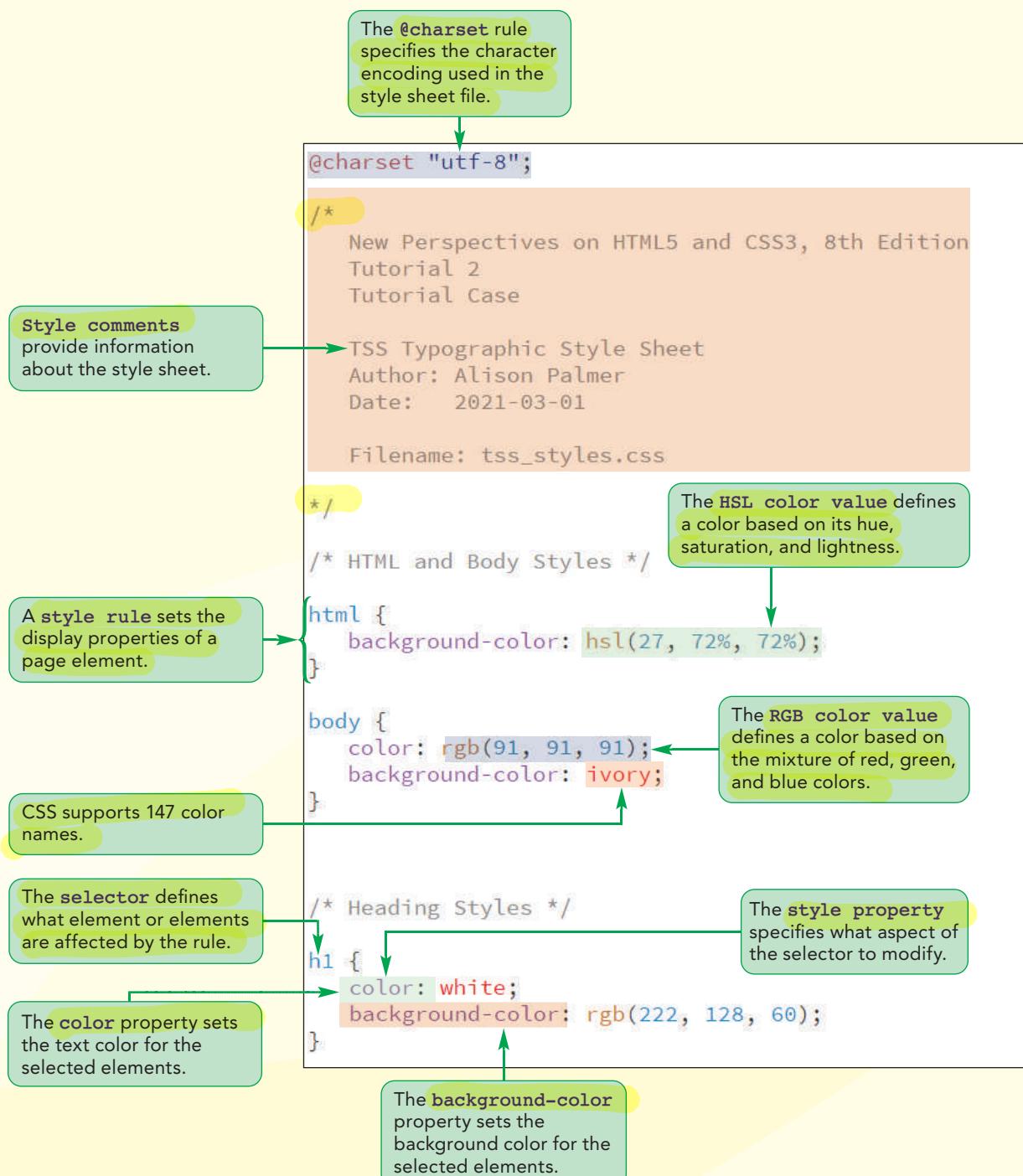
Alison Palmer runs Tri and Succeed Sports, an athletic club in Austin, Texas, that specializes in coaching men and women aspiring to compete in triathlons and other endurance sports. The center provides year-round instruction in running, swimming, cycling, and general fitness with one-on-one and group training classes. Alison has asked you to work on the company's new website.

Alison designed the original Tri and Succeed Sports website several years ago but she now feels that the site needs a makeover. She wants a new design that uses color and interesting typography to create visual interest and impact. She wants you to use CSS to help give the website a new look.

### **STARTING DATA FILES**



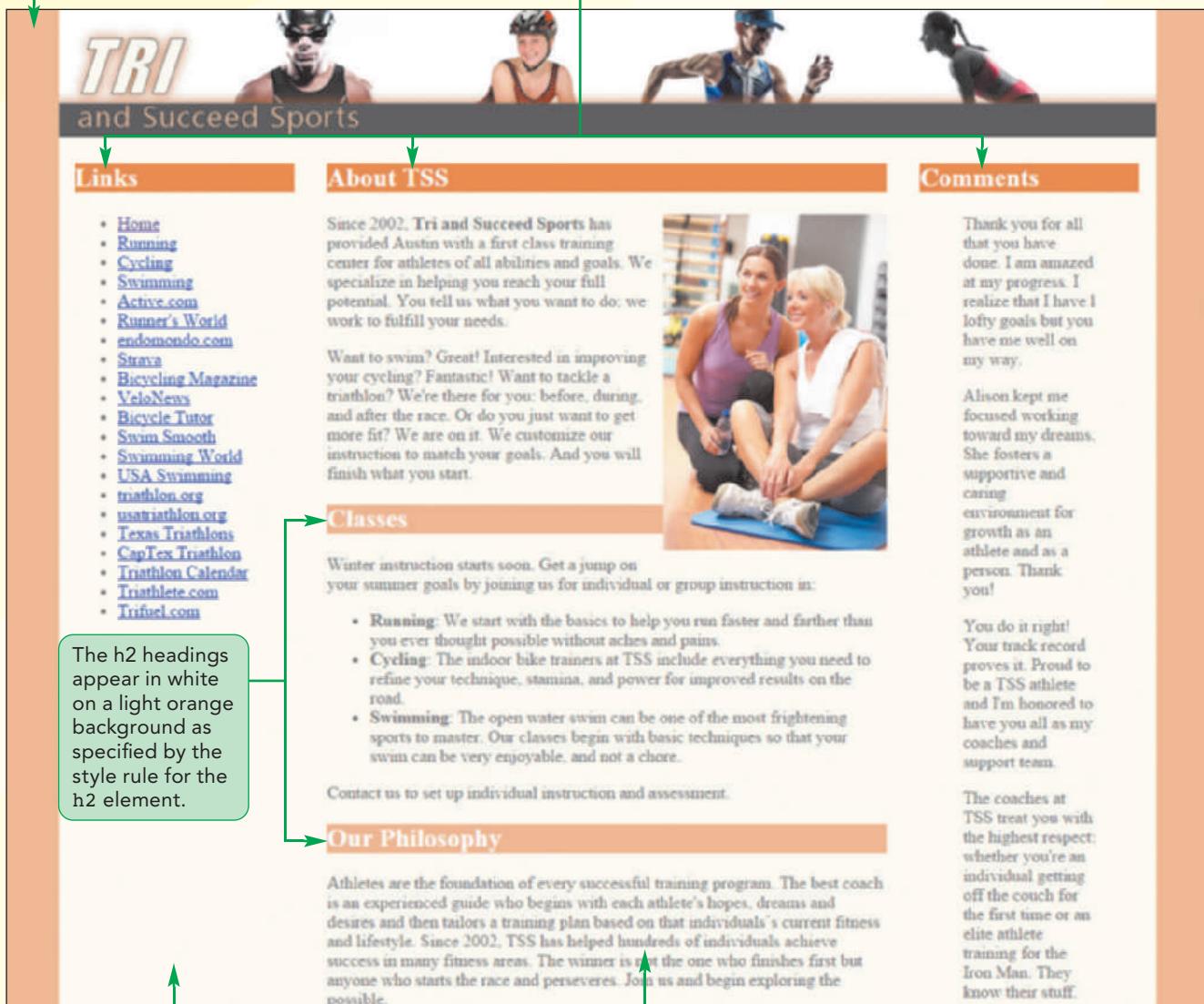
# Session 2.1 Visual Overview:



# CSS Styles and Colors

The browser window background color is set to the color value `hsl(27, 72%, 72%)` using style rule for the `html` element.

The `h1` headings appear in white on a dark orange background as specified by the style rule for the `h1` element.



The screenshot shows a website for "Tri and Succeed Sports". The header features a dark orange background with a white `h1` heading "TRI and Succeed Sports" and two images of athletes. The main content area has a light orange background. It includes sections for "Links", "About TSS", "Comments", "Classes", "Our Philosophy", and "Athletes". Arrows from the callout boxes point to specific elements in the code or the website's visual representation.

**Links**

- [Home](#)
- [Running](#)
- [Cycling](#)
- [Swimming](#)
- [Active.com](#)
- [Runner's World](#)
- [endomondo.com](#)
- [Strava](#)
- [Bicycling Magazine](#)
- [VeloNews](#)
- [Bicycle Tutor](#)
- [Swim Smooth](#)
- [Swimming World](#)
- [USA Swimming](#)
- [triathlon.org](#)
- [usatriathlon.org](#)
- [Texas Triathlons](#)
- [CapTex Triathlon](#)
- [Triathlon Calendar](#)
- [Triathlete.com](#)
- [Trifuel.com](#)

**About TSS**

Since 2002, Tri and Succeed Sports has provided Austin with a first class training center for athletes of all abilities and goals. We specialize in helping you reach your full potential. You tell us what you want to do; we work to fulfill your needs.

Want to swim? Great! Interested in improving your cycling? Fantastic! Want to tackle a triathlon? We're there for you: before, during, and after the race. Or do you just want to get more fit? We are on it. We customize our instruction to match your goals. And you will finish what you start.

**Classes**

Winter instruction starts soon. Get a jump on your summer goals by joining us for individual or group instruction in:

- Running:** We start with the basics to help you run faster and farther than you ever thought possible without aches and pains.
- Cycling:** The indoor bike trainers at TSS include everything you need to refine your technique, stamina, and power for improved results on the road.
- Swimming:** The open water swim can be one of the most frightening sports to master. Our classes begin with basic techniques so that your swim can be very enjoyable, and not a chore.

Contact us to set up individual instruction and assessment.

**Our Philosophy**

Athletes are the foundation of every successful training program. The best coach is an experienced guide who begins with each athlete's hopes, dreams and desires and then tailors a training plan based on that individual's current fitness and lifestyle. Since 2002, TSS has helped hundreds of individuals achieve success in many fitness areas. The winner is not the one who finishes first but anyone who starts the race and perseveres. Join us and begin exploring the possible.

**Comments**

Thank you for all that you have done. I am amazed at my progress. I realize that I have lofty goals but you have me well on my way.

Alison kept me focused working toward my dreams. She fosters a supportive and caring environment for growth as an athlete and as a person. Thank you!

You do it right! Your track record proves it. Proud to be a TSS athlete and I'm honored to have you all as my coaches and support team.

The coaches at TSS treat you with the highest respect: whether you're an individual getting off the couch for the first time or an elite athlete training for the Iron Man. They know their stuff.

The `h2` headings appear in white on a light orange background as specified by the style rule for the `h2` element.

Page body background color is set to ivory using the style rule for the `body` element.

Page text is set to the color value `rgb(91, 91, 91)`.

© Ysbrand Cosijn/Shutterstock.com;  
 © Charles T. Bennett/Shutterstock.com;  
 © ostill/Shutterstock.com;  
 © Monkey Business Images/Shutterstock.com

## Introducing CSS

An important principle discussed in the previous tutorial was that HTML only defines a document's content and structure, not how it should be displayed. The appearance of the page is determined by style sheets written in the Cascading Style Sheets (CSS) language. Starting with this tutorial, you'll learn how to write your own CSS style sheets.

### TIP

You can research browser support for CSS at [www.caniuse.com](http://www.caniuse.com).

The CSS language is maintained by the same World Wide Web Consortium (W3C) that defines the standards for HTML. As with HTML, CSS has gone through several versions, the latest of which is CSS Version 3, more commonly known as **CSS**. CSS is not based on a single specification but rather is built upon several **modules**, where each module is focused on a separate design topic. At the time of this writing, there were over 50 CSS modules with each module enjoying different level of browser support. The W3C continues to expand the scope of the language, which means that many new design features are still at the stage where few, if any, browsers support them.

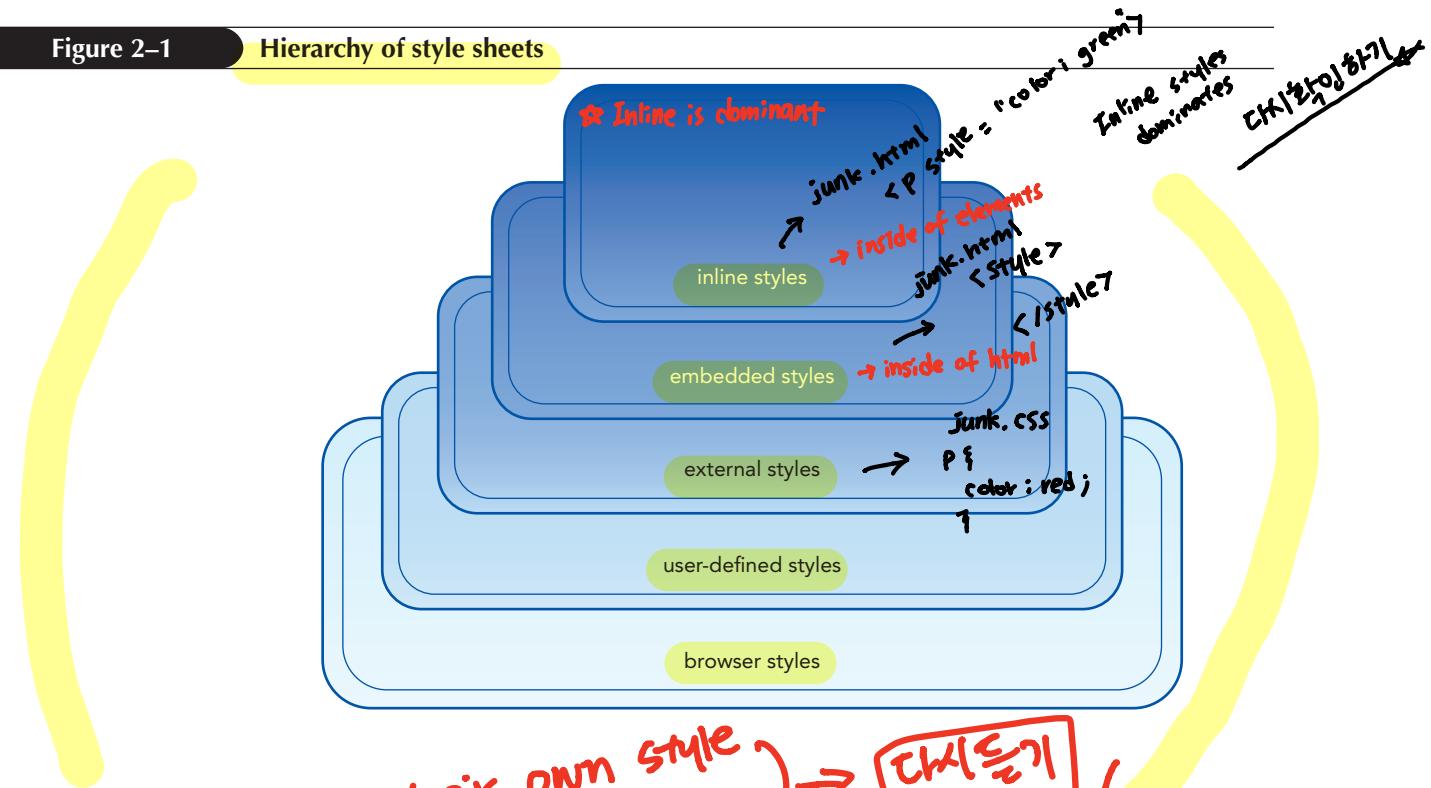
In this tutorial, you'll focus mostly on CSS features that have near-universal support among current browsers. However, you'll also examine workarounds to support older browsers and study ways to accommodate the difference between browsers in how they implement the CSS language.

## Types of Style Sheets

A website's appearance is not the product of a single style sheet; rather, it is a combination of style sheets starting from the browser style sheet and then superseded by user-defined style sheets, external style sheets, embedded style sheets, and finally, inline styles (see Figure 2–1.) Let's examine each of these style sheets in more detail.

Figure 2–1

Hierarchy of style sheets



each browser has their own style  
it shows differently  
13-14-15-16은 22%?  
→ That's it!

The first styles to be processed in rendering a website are the **browser styles** or **user agent styles**, built into the browser itself. In the absence of competing styles from other style sheets, browser styles are the ones applied to the web page.

The next styles to be processed are **user-defined styles**, created by the user within the browser. For example, a user with a visual impairment could make a website easier to read by altering the browser's default settings, displaying text in highly contrasting colors and a large font. Any user-defined style has precedence over a browser style.

User-defined styles can be superseded by **external styles**, which are styles created and placed within a CSS file and linked to the website. You used external style sheets in the last tutorial when you linked the Curbside Thai website to a collection of CSS files. As you saw in that tutorial, multiple documents can access the same style sheet, making it easier to apply a common design to an entire website.

Above external style sheets in the hierarchy are **embedded styles**, which are the styles placed within the HTML file itself. Embedded styles only apply to the HTML document in which they are created and are not accessible to other documents in the website.

Finally, at the highest order of precedence are **inline styles**, which are added as attributes of specific elements within the HTML file. The use of embedded styles and inline styles is not considered best practice because it violates one of the basic tenets of HTML: that HTML files should only describe the content and structure of the document and not the design itself.

Thus, the final appearance of the website is based on a combination of the styles from these different sources. Some of the styles might originate from the browser style sheet while others will be defined in an external style sheet or an embedded style sheet. Part of the challenge of CSS is determining how styles from these different style sheets interact to determine the page's final appearance.

## Viewing a Page Using Different Style Sheets

You'll start your work on the Tri and Succeed Sports website by viewing how the home page appears when it is rendered using only those styles in the browser style sheet.

### To view the Tri and Succeed Sports home page:

- 1. Use your editor to open the **tss\_home\_txt.html** file from the **html02** tutorial folder. Enter **your name** and **the date** in the comment section of the file and save the document as **tss\_home.html**.
- 2. Take some time to scroll through the document to become familiar with its content and structure.
- 3. Open the **tss\_home.html** page in your browser. Part of the appearance of the page is shown in Figure 2-2.

Figure 2–2

The TSS home page rendered using only the browser style sheet

heading displayed in a larger bold font

list items displayed with a solid circle bullet

hypertext displayed in blue

strong text displayed in bold

**Links**

- Home
- Running
- Cycling
- Swimming
- Active.com
- Runner's World
- endomondo.com
- Strava
- Bicycling Magazine
- VeloNews
- Bicycle Tutor
- Swim Smooth
- Swimming World
- USA Swimming
- triathlon.org
- usatriathlon.org
- Texas Triathlons
- CapTex Triathlons
- Triathlon Calendar
- Triathlete.com
- Trifuel.com

**About TSS**

Since 2002, Tri and Succeed Sports has provided Austin with a first class training center for athletes of all abilities and goals. We specialize in helping you reach your full potential. You tell us what you want to do; we work to fulfill your needs.

© Ysbrand Cosijn/Shutterstock.com; © Charles T. Bennett/Shutterstock.com; © ostill/Shutterstock.com;  
© Monkey Business Images/Shutterstock.com

**Trouble?** Depending on your browser's style sheet, your page might not exactly resemble the one shown in Figure 2–2.

The browser style sheet applies a few specific styles to the page, including adding solid circles to the navigation list items, as well as displaying hypertext in blue, headings in a large bold font, and strong text in a bold font.

However, the layout of these elements makes the page difficult to read. Alison has an external style sheet containing styles that will present this page in a more pleasing three-column layout. Link this page now to her style sheet file and then reload the document in your browser to view the impact on the page's appearance.

## To change the layout of the TSS home page:

- 1. Return to the `tss_home.html` file in your HTML editor and add the following `link` element to the head section directly after the `title` element:

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

Figure 2–3 highlights the newly added code in the document.

Figure 2–3

### Linking to the `tss_layout.css` file

`rel` attribute  
indicates that the  
file is a style sheet

```
<meta charset="utf-8" />
<meta name="keywords" content="triathlon, running, swimming, cycling" />
<title>Tri and Succeed Sports</title>
<link href="tss_layout.css" rel="stylesheet" />
</head>
```

filename of  
style sheet

- 2. Save your changes to the file and then reopen the `tss_home.html` file in your browser. Figure 2–4 shows the appearance of the page using the layout styles defined in the `tss_layout.css` file.

Figure 2–4

### The TSS home page using the `tss_layout.css` style sheet



© Ysbrand Cosijn/Shutterstock.com; © Charles T. Bennett/Shutterstock.com; © ostill/Shutterstock.com;  
© Monkey Business Images/Shutterstock.com

The `tss_layout.css` file controls the placement of the page elements but not their appearance. The colors, fonts, and other design styles are still based on the browser style sheet.

## Exploring Style Rules

If the element tag is the building block of the HTML file, then the **style rule**, which defines the styles applied to an element or group of elements, is the building block of the CSS style sheet. Style rules have the general form

```
selector {
 property1: value1;
 property2: value2;
 ...
}
```

where **selector** identifies an element or a group of elements within the document and the **property: value** pairs specify the style properties and their values applied to that element or elements. For example, the following style rule has a selector of **h1** to match all **h1** elements in the document and it has **property: value** pairs of **color: red** and **text-align: center** that tell the browser to display all **h1** headings in red and centered on the page:

```
selector
h1 { property, value
 color: red;
 text-align: center;
}
```

Selectors can also be entered as comma-separated lists as in the following style rule to display both **h1** and **h2** headings in red:

```
h1, h2 {
 color: red;
}
```

009 -:-? ③ 009  
- : -  
?

Like HTML, CSS ignores the use of white space, so you can also enter this style more compactly in a single line:

```
h1, h2 {color: red;}
```

Line breaks mean nothing  
you can write all in one line or break the line

Writing a style rule on a single line saves space, but entering each style property on a separate line often makes your code easier to read and edit. You will see both approaches used in the CSS files you encounter on the web.

## Browser Extensions

In addition to the W3C-supported style properties, most browsers supply their own extended library of style properties, known as **browser extensions**. Many of the styles that become part of the W3C specifications start as browser extensions and for older browser versions, sometimes the only way to support a particular CSS feature is through a browser extension tailored to a particular browser.

Browser extensions are identified through the use of a **vendor prefix** indicating the browser vendor that created and supports the property. Figure 2-5 lists the browser extensions you'll encounter in your work on web design.

Figure 2-5

Vendor prefixes for browser extensions

Vendor Prefix	Rendering Engine	Browsers
-khtml-	KHTML	Konqueror
-moz-	Mozilla	Firefox, Camino
-ms-	Trident	Internet Explorer
-o-	Presto	Opera, Nintendo Wii browser
-webkit-	WebKit	Android browser, Chrome, Safari

For example, one of the more recent style features added to CSS is the layout style to display content in separate columns. The number of columns is indicated using the `column-count` property. To apply this style in a way that supports both older and current browsers, you would include the browser extensions first followed by the most current CSS specification:

```
article {
 -webkit-column-count: 3;
 -moz-column-count: 3;
 column-count: 3;
}
```

In general, browsers process style properties in the order they're listed, ignoring those properties they don't recognize or support, so you always want the most current specifications listed last.

## Embedded Style Sheets

The style rule structure is also used in embedded style sheets and inline styles.

Embedded styles are inserted directly into the HTML file by adding the following `style` element to the document head

```
<style>
 style rules
</style>
```

*li { color: red; }*

```
<style>
 style rules
</style>
```

where `style rules` are the different rules you want to embed in the HTML page. For example, the following embedded style applies the same style rules described previously to make all `h1` headings in the current document appear in red and centered:

```
<style>
 h1 {
 color: red;
 text-align: center;
 }
</style>
```

Remember that, when all else is equal, the style that is loaded last has precedence over styles defined earlier. In the following code, the browser will load the embedded style sheet last, giving it precedence over the style rules in the `tss_styles.css` file.

```
<link href="tss_styles.css" rel="stylesheet" />
<style>
 style rules
</style>
```

### TIP

To avoid confusion, always place your embedded styles after any links to external style sheet files so that the embedded styles always have precedence.

If the order of the `link` and `style` elements is reversed, the styles from the `tss_styles.css` file are loaded last and given precedence.

## Inline Styles

The very last styles to be interpreted by the browser are inline styles, which are styles applied directly to specific elements using the following `style` attribute

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

where the `property: value` pairs define the styles, which are applied directly to that element. Thus, the following inline style sets the appearance of the `h1` heading to red text centered on the page:

```
<h1 style="color: red; text-align: center;">
 Tri and Succeed Sports
</h1>
```

This style applies only to this particular `h1` heading and not to any other `h1` heading on the page or in the website. The advantage of inline styles is that it is clear exactly what page element is being formatted; however, inline styles are not recommended in most cases because they make it difficult to manage your website design. For example, if you used inline styles to format all of your headings, you would have to locate all of the `h1` through `h6` elements in all of the pages within the entire website and add `style` attributes to each tag. This would be no small task on a large website containing hundreds of headings spread out among dozens of pages. Likewise, it would be a nightmare if you had to modify the design of those headings at a later date. Thus, the recommended practice is to always use external style sheets that can be applied across the entire website.

## Style Specificity and Precedence

With so many different style rules to be applied to the same document, there has to be an orderly method by which conflicts between those different rules are resolved. You've already learned that the style that is defined last has precedence, but that is not the whole story. Another important principle is that *the more specific style rule has precedence over the more general style rule*. Thus, a rule applied to a paragraph takes precedence over a rule applied to the entire page, and a rule applied to a section of text within that paragraph takes precedence over the rule for the paragraph. In the following style rules, the color of the text in all paragraphs is set to red, taking precedence over the color black applied to the rest of the text in the page:

```
p {color: red;}
body {color: black;}
```

Note that specificity is only an issue when two or more styles conflict, as in the example above. When the style rules involve different properties (such as color and size), there is no conflict and both rules are applied. If two rules have equal specificity and thus equal importance, then the one that is defined last has precedence.

## Style Inheritance

### TIP

Not all properties are inherited; for example, a `style` property that defines text color has no meaning for an inline image.

An additional factor in how style rules are interpreted is that styles are passed from a parent element to its children in a process known as **style inheritance**. Thus, the following style rule sets the color of article text to blue and that rule is passed to any paragraph, header, footer, or other element nested within an `article` element. In addition, the paragraph text within that article is centered:

```
article {color: blue;}
p {text-align: center;}
```

The final rendering of any page element is the result of styles drawn from rules across multiple style sheets and from properties passed down from one element to another within the hierarchy of page elements. These style sheets and style rules form the "cascade" of styles in Cascading Style Sheets.

## Browser Developer Tools

# F12

### TIP

In most browsers, you can quickly access information about a specific page element by right-clicking the element in the browser window and choosing Inspect Element from the pop-up menu.

If the idea of multiple style sheets and multiple style rules is intimidating, there are tools available to help you manage your styles. Most browsers include developer tools allowing the designer to view HTML code, CSS styles, and other parts of the web page. These developer tools make it easier for the designer to locate the source of a style that has been applied to a specific page element.

Each browser's developer tools are different and are constantly being updated and improved with every new browser version. However, to give you the flavor of the tools you have at your disposal, you'll examine both the HTML code and the CSS style sheet under the developer tools built into your desktop browser. Note that the figures in the steps that follow use the desktop version of the Google Chrome browser.

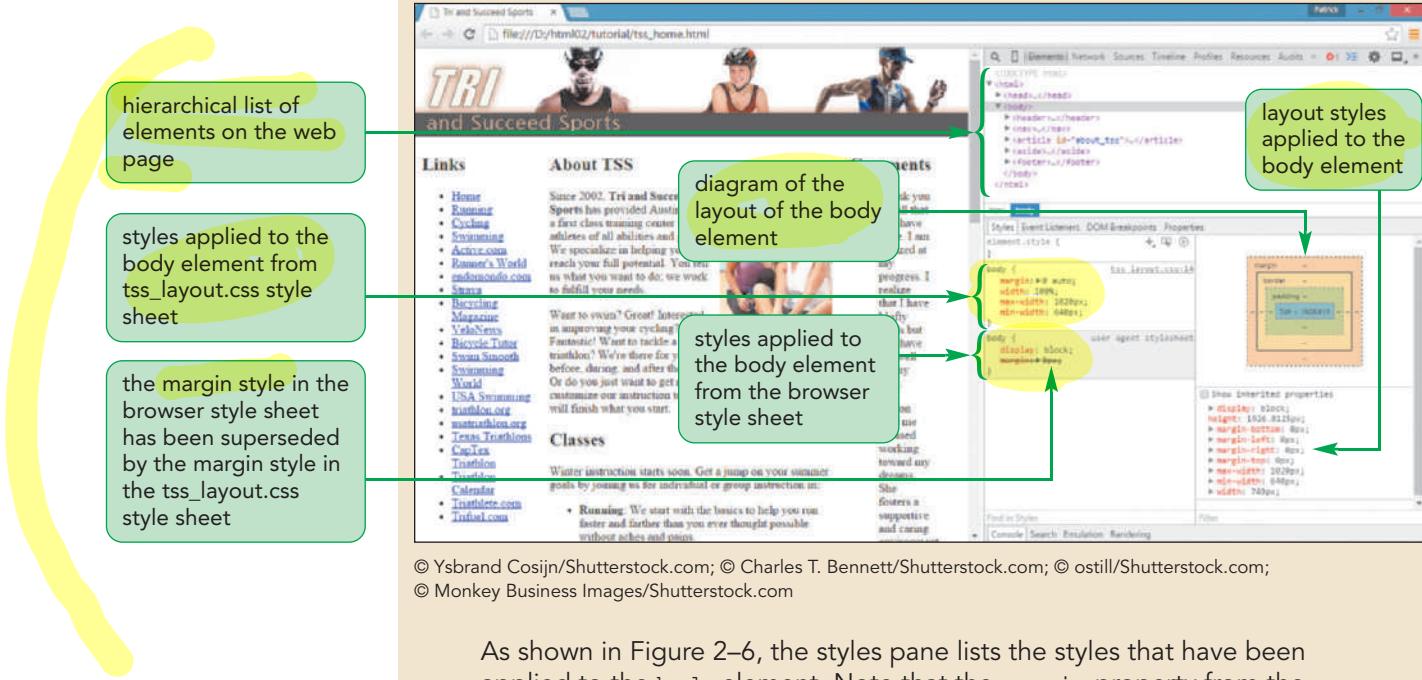
### Accessing the browser developer tools:

- 1. Return to the **tss\_home.html** file in your browser.
  - 2. Press **F12** to open the developer tools window.
- Trouble?** If pressing F12 doesn't open the developer tools, your browser might need a different keyboard combination. In Safari for the Macintosh, you can view the developer tools by pressing **ctrl+shift+l** or **command+option+l**.
- 3. From the hierarchical list of elements in the web page, click the `<body>` tag if it is not already selected.

Figure 2–6 shows the layout of panes using the developer tools under Google Chrome for the desktop.

Figure 2–6

Developer tools in Google Chrome



As shown in Figure 2–6, the styles pane lists the styles that have been applied to the `body` element. Note that the `margin` property from the browser style sheet has been crossed out, indicating that this browser style has been superseded by a style defined in the external style sheet.

**Trouble?** Every browser has a different set of developer tools and configurations. Your tools might not resemble those shown in Figure 2–6.

- 4. Take some time to explore the content and styles used in the other page elements by selecting the elements tags from the hierarchical list of elements.
- 5. Press **F12** again to close the developer tools window.

**Trouble?** In Safari, you can close the developer tools by pressing **ctrl+shift+l** or by **command+option+l**.

In this and future tutorials, you may find that your browser's developer tools are a great aid to working through your website designs. Most developer tools allow the user to insert new style rules in order to view their immediate impact on the page's appearance; however, these modifications are only applied during the current session and are not saved. So, once you find a setting that you want to use, you must enter it in the appropriate style sheet for it to take effect permanently.

## INSIGHT

### Defining an **!important** Style

You can override the style cascade by marking a particular property with the following **!important** keyword:

```
property: value !important;
```

The following style rule sets the color of all **h1** headings to orange; and because this property is marked as **important**, it takes precedence over any conflicting styles found in other style sheets.

```
h1 {color: orange !important;}
```

The **!important** keyword is most often used in user-defined style sheets in which the user needs to substitute his or her own styles in place of the designer's. For example, a visually impaired user might need to have text displayed in a large font with highly contrasting colors. In general, designers should not use the **!important** keyword because it interferes with the cascade order built into the CSS language.

## Creating a Style Sheet

Now that you've reviewed some history and concepts behind style sheets, you'll start creating your own. You should usually begin your style sheets with comments that document the purpose of the style sheet and provide information about who created the document and when.

### Writing Style Comments

Style sheet comments are entered as

```
/*
comment
*/
```

where *comment* is the text of the comment. Because CSS ignores the presence of white space, you can insert your comments on a single line to save space as:

```
/* comment */
```

Create a style sheet file now, placing a comment with your name and the current date at the top of the file.

### Writing a Style Comment:

- 1. Use your editor to open the **tss\_styles\_txt.css** file from the **html02 ▶ tutorial** folder.
- 2. Within the comment section at the top of the file, enter **your name** following the **Author: comment** and **the date** following the **Date: comment**.
- 3. Save the file as **tss\_styles.css**.
- 4. Return to the **tss\_home.html** file in your HTML editor and add the following **link** element directly before the closing **</head>** tag.

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

- 5. Close the **tss\_home.html** file, saving your changes.

### Defining the Character Encoding

As with HTML files, it is a good idea in every CSS document to define the character encoding used in the file. In CSS, you accomplish this using the following **@charset** rule

```
@charset "encoding";
```

where *encoding* defines the character encoding used in the file. Add the **@charset** rule to the **tss\_styles.css** style sheet file now, specifying that the **UTF-8** character set is used in the CSS code.

### To indicate the character encoding:

- 1. Return to the **tss\_styles.css** file in your editor.
- 2. Directly above the initial comment section, insert the line: **@charset "utf-8";**.

Figure 2-7 highlights the new code in the style sheet.

**Figure 2-7** Adding the **@charset** rule and style comments

the charset rule defines the character encoding used in the style sheet

@charset "utf-8";

/\*  
 New Perspectives on HTML5 and CSS3, 8th Edition  
 Tutorial 2  
 Tutorial Case  
  
 TSS Typographic Style Sheet  
 Author: Alison Palmer  
 Date: 2021-03-01

author name and current date

Filename: tss\_styles.css

\*/

Note that only one @charset rule should appear in a style sheet and it should always precede any other characters, including comments.

- 3. Save your changes to the file.

## Importing Style Sheets

### TIP

The @import statement must always come before any other style rules in the style sheet.

The @charset rule is an example of a **CSS at-rule**, which is a rule used to send directives to the browser indicating how the contents of the CSS file should be interpreted and parsed. Another at-rule is the following **@import** used to import the contents of a style sheet file

```
@import url(url);
```

where *url* is the URL of an external style sheet file.

The @import is used to combine style rules from several style sheets into a single file. For example, an online store might have one style sheet named basic.css containing all of the basic styles used in every web page and another style sheet named sales.css containing styles used with merchandise-related pages. The following code imports styles from both files:

```
@import url(basic.css);
@import url(sales.css);
```

Using multiple @import rules in a CSS file has the same impact as adding multiple link elements to the HTML file. One advantage of the @import rule is that it simplifies your HTML code by placing the decision about which style sheets to include and exclude in the CSS file rather than in the HTML file.

## Working with Color in CSS

The first part of your style sheet for the Tri and Succeed Sports website will focus on color. If you've worked with graphics software, you've probably made your color selections using a graphical interface where you can see your color options. Specifying color with CSS is somewhat less intuitive because CSS is a text-based language and requires colors to be defined in textual terms. This is done through a color name or a color value.

### Color Names

### TIP

You can view the complete list of CSS color names by opening the demo\_color\_names.html file in the html02 ▶ demo folder.

You've already seen from previous code examples that you can set the color of page text using the color property along with a color name such as red, blue, or black. CSS supports 147 color names covering common names such as red, green, and yellow to more exotic colors such as ivory, orange, crimson, khaki, and brown.



### Written Communication: Communicating in Color

Humans are born to respond to color. Studies have shown that infants as young as two months prefer bright colors with strong contrast to drab colors with little contrast, and market research for clothing often focuses on what colors are “in” and what colors are passé.

Your color choices can impact the way your website is received so you want to choose a color scheme that is tailored to the personality and interests of your target audience. Color can evoke an emotional response and is associated with particular feelings or concepts, such as

- *red*—assertive, powerful, sexy, dangerous
- *pink*—innocent, romantic, feminine
- *black*—strong, classic, stylish
- *gray*—business-like, detached
- *yellow*—warm, cheerful, optimistic
- *blue*—consoling, serene, quiet
- *orange*—friendly, vigorous, inviting
- *white*—clean, pure, straightforward, innocent

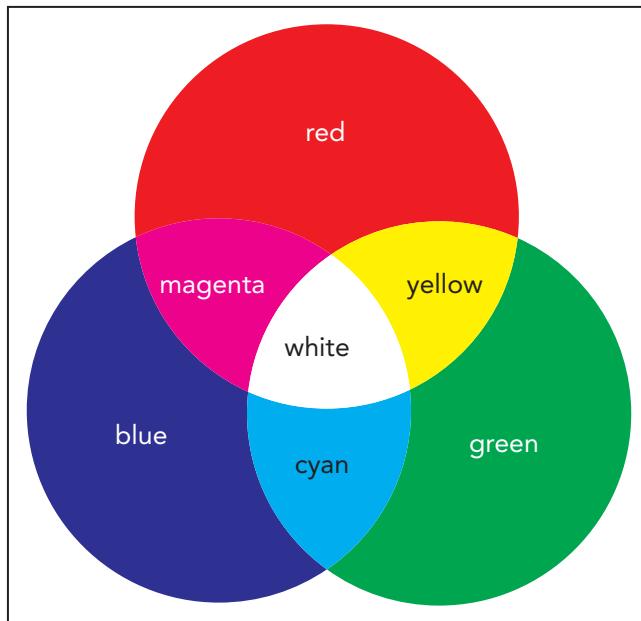
If your website will be used internationally, you need to be aware of how cultural differences can affect your audience’s response to color. For instance, white, which is associated with innocence in Western cultures, is the color of mourning in China; yellow, which is considered a bright, cheerful color in the West, represents spirituality in Buddhist countries.

### RGB Color Values

Because a palette of 147 color names is extremely limited for graphic design and color names can be constricting (how do you name a color that is slightly redder than ivory with a tinge of blue?), CSS also supports **color values**, in which the color is given by an exact numeric representation. CSS supports two types of color values: RGB values and HSL values.

RGB color values are based on classical color theory in which all colors are determined by adding three primary colors—red, green, and blue—at different levels of intensity. For example, adding all three primary colors at maximum intensity produces the color white, while adding any two of the three primary colors at maximum intensity produces the trio of complementary colors—yellow, magenta, and cyan (see Figure 2–8).

Figure 2-8 Color addition in the RGB color model



Varying the intensity of the three primary colors extends the palette to other colors. Orange, for example, is created from a high intensity of red, a moderate intensity of green, and a total absence of blue. CSS represents these intensities mathematically as a set of numbers called an **RGB triplet**, which has the format

`rgb(red, green, blue)`

#### TRY IT

You can explore the RGB color values using the `demo_rgb.html` file in the `html02 ▶ demo` folder.

where `red`, `green`, and `blue` are the intensities of the red, green, and blue components of the color. Intensities range from 0 (absence of color) to 255 (maximum intensity); thus, the color white has the value `rgb(255, 255, 255)`, indicating that red, green, and blue are mixed equally at the highest intensity, and orange is represented by `rgb(255, 165, 0)`. RGB triplets can describe 256<sup>3</sup> (16.7 million) possible colors, which is a greater number of colors than the human eye can distinguish.

RGB values are sometimes expressed as hexadecimal numbers where a **hexadecimal number** is a number expressed in the base 16 numbering system rather than in the commonly used base 10 system. In base 10 counting, numeric values are expressed using combinations of 10 characters (0 through 9). Hexadecimal numbering includes these ten numeric characters and six extra characters: A (for 10), B (for 11), C (for 12), D (for 13), E (for 14), and F (for 15). For values above 15, you use a combination of those 16 characters. For example, the number 16 has a hexadecimal representation of 10, and a value of 255 has a hexadecimal representation of FF. The style value for color represented as a hexadecimal number has the form

`#redgreenblue`

where `red`, `green`, and `blue` are the hexadecimal values of the red, green, and blue components. Therefore, the color yellow could be represented either by the RGB triplet

`rgb(255, 255, 0)`

or more compactly as the hexadecimal

`#FFFF00`

If the colors are different, (if it happens)  
the last one is applied

p {  
color: rgb(255, 255, 0);  
color: yellow;  
color: green;}

p {  
color: rgb(0, 0, 255);  
color: #0000FF;  
color: blue;}

all same

Most HTML editors and graphic programs provide color picking tools that allow the user to choose a color and then copy and paste the RGB or hexadecimal color value. Hexadecimal color values have the advantage of creating smaller style sheets, which can be loaded faster—an important consideration for mobile devices. However, for others viewing and studying your style sheet code, they are more difficult to interpret than RGB values.

Finally you can enter each component value as a percentage, with 100% representing the highest intensity. In this form, you would specify the color orange with the following values

`rgb(100%, 65%, 0%)`

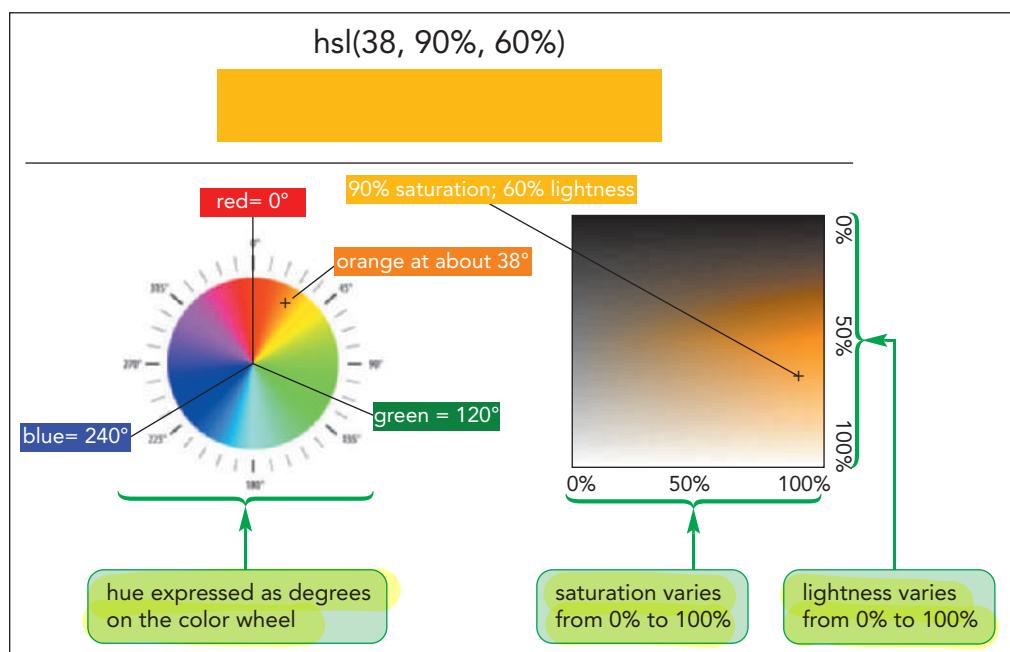
which is equivalent to the `rgb(255, 165, 0)` value described above.

## HSL Color Values

HSL color values are based on a color model in which each color is determined by its hue, saturation, and lightness. **Hue** is the tint of the color and is usually represented by a direction on a color wheel. Hue values range from 0° up to 360°, where 0° matches the location of red on the color wheel, 120° matches green, and 240° matches blue. **Saturation** measures the intensity of the chosen color and ranges from 0% (no color) up to 100% (full color). Finally, **lightness** measures the brightness of the color and ranges from 0% (black) up to 100% (white). Figure 2–9 shows how setting the hue to 38°, the saturation to 90%, and the lightness to 60% results in a medium shade of orange.

Figure 2–9

Defining the color orange under the HSL color model



Color values using the HSL model are described in CSS using

`hsl(hue, saturation, lightness)`

where `hue` is the tint of the color in degrees, `saturation` is the intensity in percent, and `lightness` is the brightness in percent of the color. Thus, a medium orange color would be represented as

`hsl(38, 90%, 60%)`

### TRY IT

You can explore the HSL color values using the `demo_hsl.html` file in the `html02 ▶ demo` folder.

Graphic designers consider HSL easier to use because it allows them to set the initial color based on hue and then fine-tune the saturation and lightness values. This is more difficult in the RGB model because you have to balance three completely different colors to achieve the right mix. For example, the RGB equivalent to the color orange in Figure 2–9 would be the color value `rgb(245, 177, 61)`; however, it's not immediately apparent why that mixture of red, green, and blue would result in that particular shade of orange.

## Defining Semi-Opaque Colors

make the color  
transparent from 0 to 100 %

Colors can also be semi-opaque by setting the color's **opacity**, which defines how solid the color appears. The color's opacity can be specified using either of the following `rgba` and `hsla` properties

`rgba(red, green, blue, opacity)`  
`hsla(hue, saturation, lightness, opacity)`

### TRY IT

You can explore the RGBA and HSLA color values using the `demo_rgba.html` and `demo_hsla.html` files in the `html02 ▶ demo` folder.

where `opacity` is the opacity of the color ranging from 0 (completely transparent) up to 1.0 (completely opaque). For example, the following style property uses the HSL color model to define a medium orange color with an opacity of 0.7:

`hsla(38, 90%, 60%, 0.7)`

The final appearance of a semi-opaque color is influenced by the colors behind it on the page. Displayed against a white background, a medium orange color would appear in a lighter shade of orange because the orange will appear mixed with the background white.

On the other hand, the same orange color displayed on a black background would appear as a darker shade of orange. The advantage of using semi-transparent colors is that it makes it easier to create a color theme in which similarly tinted colors blend with other colors on the page.

## Setting Text and Background Colors

Now that you've studied how CSS works with colors, you can start applying color to some of the elements displayed on the Tri and Succeed Sports website. CSS supports the following styles to define both the text and background color for each element on your page

`color: color;`  
`background-color: color;`

where `color` is either a color value or a color name.

Alison wants to use an HSL color value (27, 72%, 72%) to set the background of the document to orange and she would like the text of the home page to appear in a medium gray color on an ivory background. The style rules to modify the appearance of these document elements are

```
html {
 background-color: hsl(27, 72%, 72%);
}
body {
 color: rgb(91, 91, 91);
 background-color: ivory;
}
```

The `html` selector in this code selects the entire HTML document so that any part of the browser window background that is not within the page body will be displayed using the HSL color (27, 72%, 72%).

Within the page body, Alison wants the `h1` and `h2` headings displayed in white text on dark and lighter orange colors using the RGB color values (222, 128, 60) and (235, 177, 131) respectively. The style rules are

```
h1 {
 color: white;
 background-color: rgb(222, 128, 60);
}
h2 {
 color: white;
 background-color: rgb(235, 177, 131);
}
```

**REFERENCE**

### Setting Text and Background Color

- To set the text color of an element, use the following property  
`color: color;`
- To set the background color of an element, use the following property  
`background-color: color;`  
where `color` is a color name or a color value.

Next, add style rules for text and background colors to the `tss_styles.css` file.

Saturation and lightness values in an hsl color value must be expressed as percentages.

### To define background and text colors:

1. Add the following code within the HTML and Body Styles section:

```
html {
 background-color: hsl(27, 72%, 72%);
}

body {
 color: rgb(91, 91, 91);
 background-color: ivory;
}
```

**TIP**

Almost 8% of all men and 0.5% of all women have some sort of color blindness. Because red/green color blindness is the most common type of color impairment, you should avoid using red text on a green background and vice versa.

2. Add the following style rules within the Heading Styles section:

```

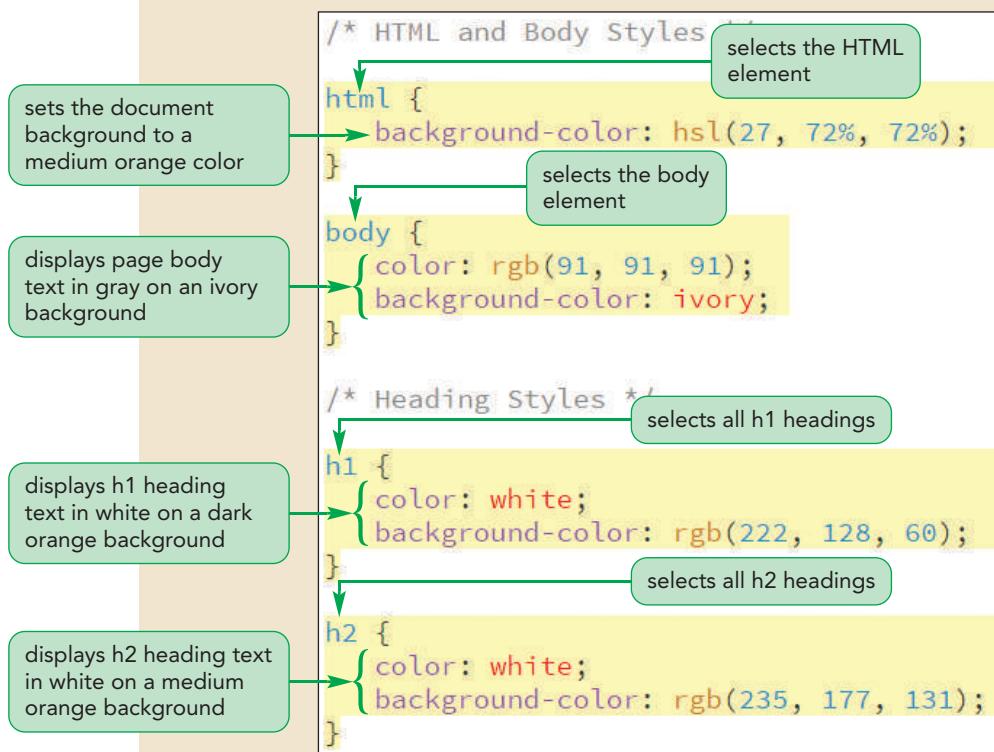
h1 {
 color: white;
 background-color: rgb(222, 128, 60);
}

h2 {
 color: white;
 background-color: rgb(235, 177, 131);
}

```

Figure 2–10 highlights the new style rules.

**Figure 2–10** Adding text and background colors



3. Save your changes to the file and then reload the `tss_home.html` file in your browser. Figure 2–11 shows the appearance of the page under the new styles.

Figure 2–11

## Text and background colors in the web page

white h1 heading text on a dark orange background

browser window background is medium orange

white h2 heading text on a light orange background

Links

- Home
- Running
- Cycling
- Swimming
- Active.com
- Runner's World
- Endomondo.com
- Strava
- Bicycling Magazine
- VeloNews
- Bicycle Tutor
- Swim Smooth
- Swimming World
- USA Swimming
- triathlon.org
- usatriathlon.org
- Texas Triathlons
- CapTex Triathlon
- Triathlon Calendar
- Triathlete.com

About TSS

Since 2002, Tri and Succeed Sports has provided Austin with a first class training center for athletes of all abilities and goals. We specialize in helping you reach your full potential. You tell us what you want to do; we work to fulfill your needs.

Want to swim? Great! Interested in improving your cycling? Fantastic! Want to tackle a triathlon? We're there for you: before, during, and after the race. Or do you just want to get more fit? We are on it. We customize our instruction to match your goals. And you will finish what you start.

Classes

Winter instruction starts soon. Get a jump on your summer goals by joining us for individual or group instruction in:

Comments

Thank you for all that you have done. I am amazed at my progress. I realize that I have lofty goals but you have me well on my way.

Alison kept me focused working toward my dreams. She fosters a supportive and caring environment for growth as an athlete and as a person. Thank you!

© Ysbrand Cosijn/Shutterstock.com;  
 © Charles T. Bennett/Shutterstock.com;  
 © ostill/Shutterstock.com;  
 © Monkey Business Images/Shutterstock.com

page body style shows gray text on an ivory background

**Trouble?** The text of hypertext links in the left column is blue, using the default browser styles applied to hypertext links. You'll modify these colors later in the tutorial.



PROSKILLS

### Problem Solving: Choosing a Color Scheme

One of the worst things you can do to your website is to associate interesting and useful content with jarring and disagreeable color. Many designers prefer the HSL color system because it makes it easier to select visually pleasing color schemes. The following are some basic color schemes you may want to apply to websites you design:

- **monochrome**—a single hue with varying values for saturation and lightness; this color scheme is easy to manage but is not as vibrant as other designs
- **complementary**—two hues separated by 180° on the color wheel; this color scheme is the most vibrant and offers the highest contrast and visual interest, but it can be misused and might distract users from the page content
- **triad**—three hues separated by 120° on the color wheel; this color scheme provides the same opportunity for pleasing color contrasts as a complementary design, but it might not be as visibly striking
- **tetrad**—four hues separated by 90° on the color wheel; perhaps the richest of all color schemes, it is also the hardest one in which to achieve color balance
- **analogic**—three hues close to one another on the color wheel in which one color is the dominant color and the other two are supporting colors used only for highlights and nuance; this scheme lacks color contrasts and is not as vibrant as other color schemes

Once you have selected a color design and the main hues, you then vary those colors by altering the saturation and lightness. One of the great advantages of style sheets is that you can quickly modify your color design choices and view the impact of those changes on your page content.

TRY IT  
You can explore color schemes with the `demo_anal.html`, `demo_comp.html`, `demo_mono.html`, `demo_split.html`, and `demo_triad.html` files in the `html02 ▶ demo` folder.

## Employing Progressive Enhancement

The HSL color you used for the `html` selector was introduced with CSS and thus it is not supported in very old browsers. If this is a concern, you can insert the older style properties first followed by the newer standards. For example, the following style rule sets the background color of the `html` element to a lighter orange using the RGB value first, and then the equivalent HSL value.

```
html {
 background-color: rgb(235, 177, 131);
 background-color: hsl(27, 72%, 72%);
}
```

Old browsers that don't recognize the HSL color value will ignore it and use the RGB value, while browsers that recognize both values will use the one that is defined last, which in this case is the HSL value. This is an example of a technique known as **progressive enhancement**, which places code conforming to older standards before newer properties, providing support for old browsers but still allowing newer standards and techniques to be used by the browsers that support them.

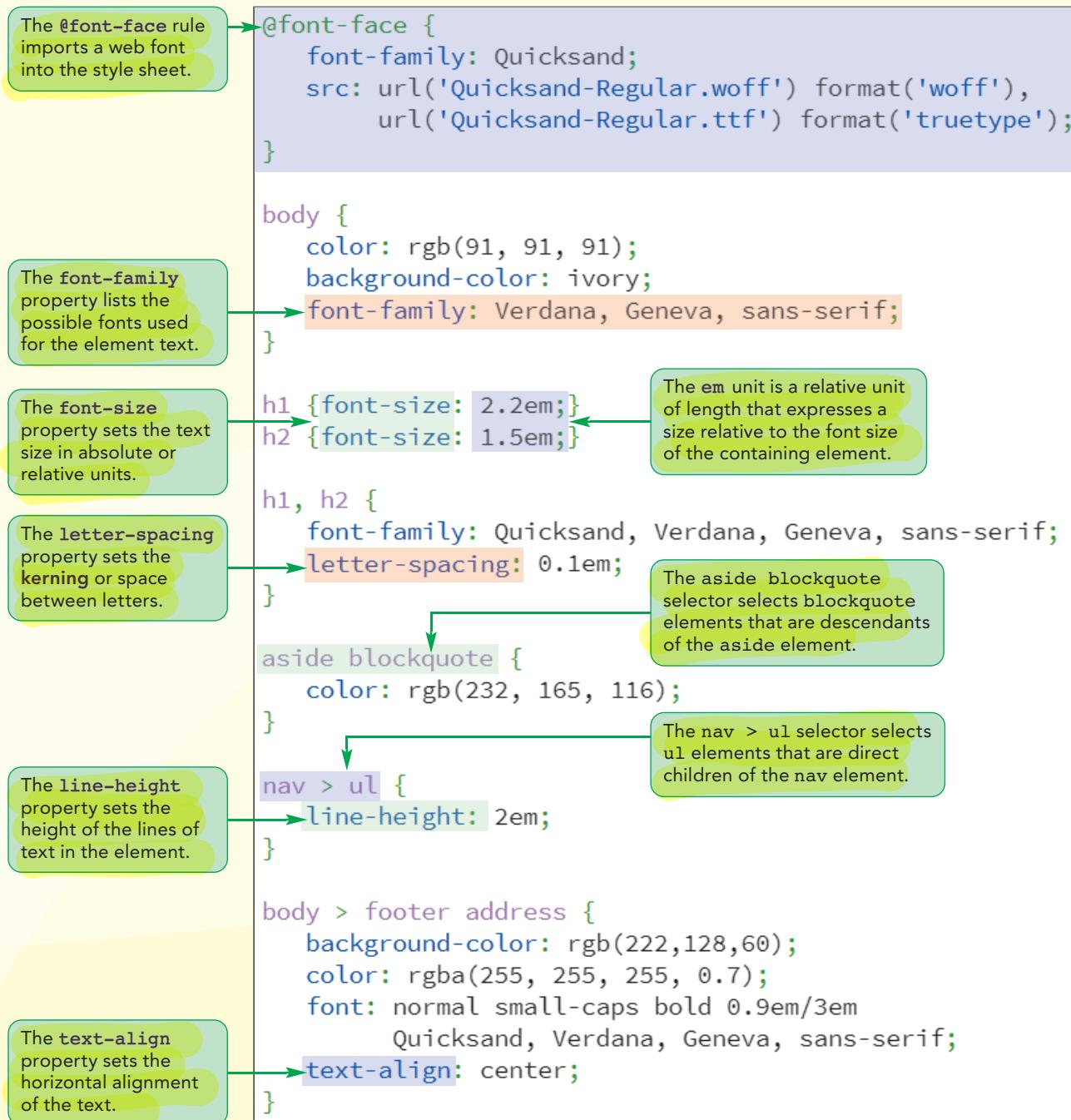
You show Alison the work you've done on colors. She's pleased with the ease of using CSS to modify the design and appearance of elements on the Tri and Succeed Sports website. In the next session, you'll continue to explore CSS styles, focusing on text styles.

## REVIEW

## Session 2.1 Quick Check

1. Which styles are applied directly to elements within an HTML file?
  - browser styles
  - inline styles *→ <h1 style="..."> </h1>*
  - embedded styles *→ <style> ... </style>*
  - external styles *→ CSS*
2. What keyword do you add to a style property to override style precedence and style inheritance?
  - cascade!
  - use!
  - important** *!important*
  - import!
3. CSS comments are entered as:
  - /\* comment \*/*
  - ! comment !*
  - /\* comment*
  - <!-- comment ---> → HTML*
4. Which of the following values is used to represent the color red?
  - rgb(10, 0, 0)
  - rgb(1000, 0, 0)
  - rgb(255, 0, 0)**
  - rgb(0, 255, 0)
5. What is the hue associated with the color value hsl(90, 100%, 50%)?
  - 90**
  - 100%
  - 50%
  - Cannot be determined by the color value
6. What is the vendor prefix associated with the Google Chrome browser?
  - chrome-
  - google-
  - moz-
  - webkit-**
7. What is the HSL color value for red displayed with the highest saturation and lightness and with 50% transparency?
  - hsl(0, 100%, 100%, 0.5)
  - hsl(255, 100%, 100%, 0.5)
  - hsla(0, 100%, 100%, 0.5)**
  - hsla(0.5, 0, 100%, 100%)
8. What is the style to display h1 headings in green text on a yellow background?
  - h1 {color: green; background-color: yellow;}**
  - h1 {color: green; bgcolor: yellow;}
  - h1 {text: green; background: yellow;}
  - h1 {textColor: green; backgroundColor: yellow;}
9. Which of the following matches the color blue?
  - rgb(0, 0, 255)**
  - hsl(240, 100%, 50%);
  - rgba(0, 0, 255, 1)
  - All of the above

## Session 2.2 Visual Overview:



© Monkey Business Images/Shutterstock.com

# CSS Typography

The h1 heading is displayed in the Quicksand font with a font size of 2.2em and letter spacing of 0.1em.

## Links

- [Home](#)
  - [Running](#)
  - [Cycling](#)
  - [Swimming](#)
  - [Active.com](#)
  - [Runner's World](#)
  - [endomondo.com](#)
  - [Strava](#)
  - [Bicycling Magazine](#)
  - [VeloNews](#)
  - [Bicycle Tutor](#)
  - [Swim Smooth](#)
  - [Swimming World](#)
  - [USA Swimming](#)
  - [triathlon.org](#)
  - [usatriathlon.org](#)
  - [Texas Triathlons](#)
  - [CapTex Triathlon](#)
  - [Triathlon Calendar](#)
  - [Triathlete.com](#)
  - [Trifuel.com](#)

Navigation list is double-spaced with a line height of 2em.

## About TSS

Since 2002, **Tri and Succeed Sports** has provided Austin with a first class training center for athletes of all abilities and goals. We specialize in helping you reach your full potential. You tell us what you want to do; we work to fulfill your needs.

Want to swim? Great! Interested in improving your cycling? Fantastic! Want to tackle a triathlon? We're there for you: before, during, and after the race. Or do you just want to get more fit? We are on it. We customize our instruction to match your goals. And you will finish what you start.

## Classes

Winter instruction starts soon. Get a jump on your summer goals by joining us for individual or group instruction in:

- **Running:** We start with the basics to help you run faster and farther than you ever thought possible without aches and pains.
  - **Cycling:** The indoor bike trainers at TSS include everything you need to refine your technique, stamina, and power for improved results on the road.
  - **Swimming:** The open water swim can be one of the most frightening sports to master. Our classes begin with basic techniques so that your swim can be very enjoyable, and not a chore.

Contact us to set up individual instruction and assessment.

## Our Philosophy

Athletes are the foundation of every successful training program. The best coach is an experienced guide who begins with each athlete's hopes, dreams and desires and then tailors a training plan based on that individual's current fitness and lifestyle. Since 2002, TSS has helped hundreds of individuals achieve success in many fitness areas. The winner is not the one who finishes first but anyone who starts the race and perseveres. Join us and begin exploring the possible.

Page footer is centered and displayed in small caps as specified by the `body > footer address` style rule.

The h2 headings are displayed in the Quicksand font with a font size of 1.5em and letter spacing of 0.1em.

Copyright 2021 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s). Editorial review has determined that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

## Exploring Selector Patterns

The following style rule matches every `h1` element in the HTML document, regardless of its location within the page:

```
h1 {
 color: red;
}
```

Often, however, you will want your style rules to apply to elements only placed within specific locations, such as `h1` headings found within articles but not anywhere else. To direct a style rule to an element at a specific location use **selector patterns** to match only those page elements that match a specified pattern.

### Contextual Selectors

The first selector pattern you'll examine is a **contextual selector**, which specifies the context under which a particular page element is used. Context is based on the hierarchical structure of the document, which involves the relationships between a **parent element** containing one or more **child elements** and within those child elements several levels of **descendant elements**. A contextual selector relating a parent element to its descendants has the following pattern

```
parent descendant { styles }
```

where `parent` is a parent element, `descendant` is a descendant of that parent, and `styles` are styles applied to the descendant element. For example, the following style rule sets the text color of `h1` headings to red but only when those headings are nested somewhere within the `header` element:

```
header h1 {
 color: red;
}
```

As shown in the following code, the descendant element does not have to be a direct child of the parent; in fact, it can appear several levels below the parent in the hierarchy. This means that the above style rule matches the `h1` element in the following HTML code:

```
<header>
 <div>
 <h1>Tri and Succeed Sports</h1>
 </div>
</header>
```

The `h1` element is a direct child of the `div` element; but, because it is still a descendant of the `header` element, the style rule still applies.

Contextual selectors follow the general rule discussed in the last session; that is, the more specific style is applied in preference to the more general rule. For instance, the following style rules would result in `h1` headings within the `section` element being displayed in red while all other `h1` headings would appear in blue:

```
section h1 {color: red;}
h1 {color: blue;}
```

Figure 2–12 describes some of the other contextual selectors supported by CSS.

Figure 2-12 Contextual selectors

Selector	Description
<code>*</code>	Matches any element
<code>elem</code>	Matches the element <code>elem</code> located anywhere in the document
<code>elem1, elem2, ...</code>	Matches any of the elements <code>elem1</code> , <code>elem2</code> , etc.
<code>parent descendant</code>	Matches the <code>descendant</code> element that is nested within the <code>parent</code> element at some level
<code>parent &gt; child</code>	Matches the <code>child</code> element that is a child of the <code>parent</code> element
<code>elem1 + elem2</code>	Matches <code>elem2</code> that is immediately preceded by the sibling element <code>elem1</code>
<code>elem1 ~ elem2</code>	Matches <code>elem2</code> that follows the sibling element <code>elem1</code>

To match any element, use the **wildcard selector** with the `*` character. For example, the following style rule matches every child of the `article` element, setting the text color to blue:

```
article > * {color: blue;}
```

**Sibling selectors** select elements based on the elements that are adjacent to them in the document hierarchy. The following style rule uses the `+` symbol to select the `h2` element, but only if it is immediately preceded by an `h1` element:

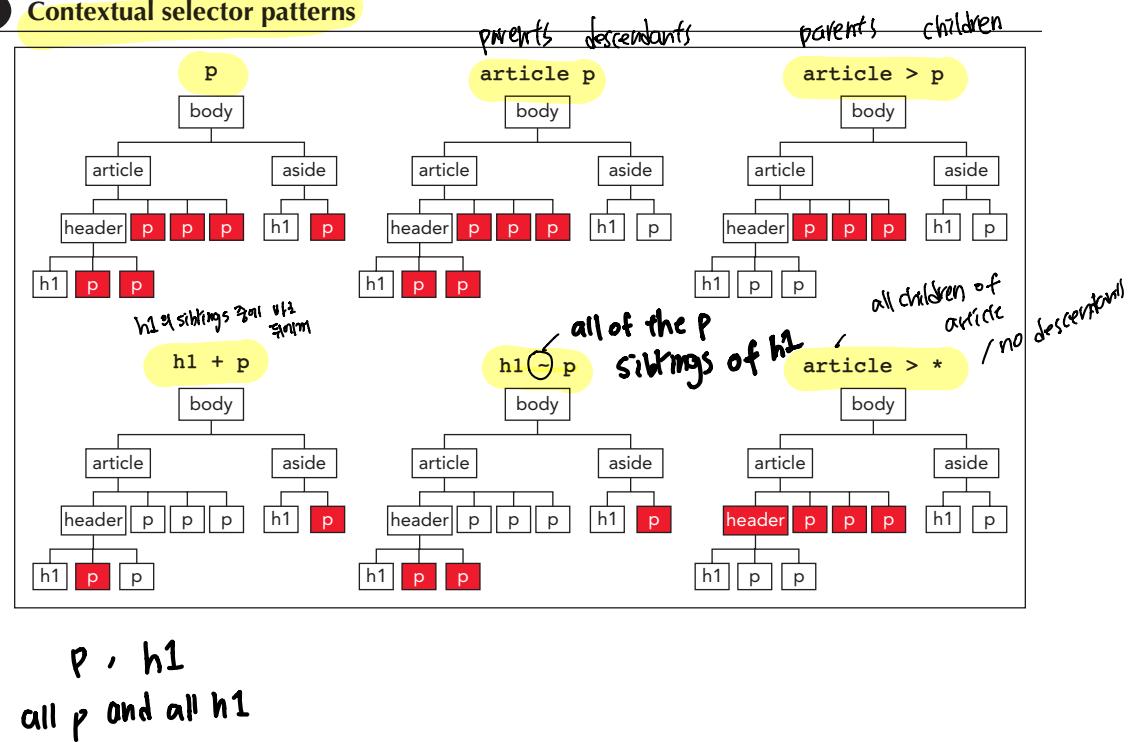
```
h1 + h2 {color: blue;}
```

On the other hand, the following style rule uses the `~` symbol to select any `h2` element that is preceded (but, not necessarily immediately) by an `h1` element:

```
h1 ~ h2 {color: blue;}
```

Figure 2-13 provides additional examples of selectors and highlights in red those elements in the document that would be selected by the specified selector.

Figure 2-13 Contextual selector patterns



Remember that, because of style inheritance, any style applied to an element is passed down the document tree. Thus, a style applied to a `header` element is automatically passed down to elements contained within that header unless that style conflicts with a more specific style.

## REFERENCE

### Using Contextual Selectors

- To select all elements, use the `*` selector.
- To select a single element, use the `elem` selector, where `elem` is the name of the element.
- To select a descendant element, use the `parent descendant` selector where `parent` is a parent element and `descendant` is an element nested within the parent at some lower level.
- To select a child element, use the `parent > child` selector.
- To select a sibling element, `elem2`, that directly follows `elem1`, use the `elem1 + elem2` selector.
- To select a sibling element, `elem2`, that follows, but not necessarily directly `elem1`, use the `elem1 ~ elem2` selector.

Now, you'll create a style rule to change the text color of the customer testimonials on the Tri and Succeed Sports home page to a dark orange using the RGB color value `rgb(232, 165, 116)`. You'll use a contextual selector to apply the style rule only to block quotes that are descendants of the `aside` element.

### To create style rule with a contextual selector:

1. If you took a break after the previous session, make sure the `tss_styles.css` file is open in your editor.
2. Within the Aside and Blockquote Styles section, insert the following style rule:

```
aside blockquote {
 color: rgb(232, 165, 116);
}
```

Figure 2–14 highlights the new style rule for the `blockquote` element.

Figure 2–14

### Setting the text color of block quotes

```
/* Aside and Blockquote Styles */
aside blockquote {
 color: rgb(232, 165, 116);
}
```

style applies to block quotes nested within an aside element

`/* Aside and Blockquote Styles */`

`aside blockquote {`

`color: rgb(232, 165, 116);`

`}`

sets the text color to dark orange

3. Save your changes to the file and then reload the `tss_home.html` file in your browser. Verify that the text of the customer quotes appears in orange.

Copyright 2021 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s). Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

## Attribute Selectors

Contextual selectors can also be based on attributes and attribute values within elements. Two attributes, `id` and `class`, are often key in targeting styles to specific elements. Recall that the `id` attribute is used to identify specific elements within the document. To apply a style to an element based on its `id`, you use either the selector

`#id`

or the selector

`elem#id`

where `id` is the value of the `id` attribute and `elem` is the name of the element. Because `ids` are supposed to be unique, either form is acceptable but including the element name removes any confusion about the location of the selector. For example, the selector for the following `h1` heading from the HTML file

```
<h1 id="title">Tri and Succeed Sports</h1>
```

can be entered as either `#title` or `h1#title` in your CSS style sheet.

Because no two elements can share the same `ID`, HTML uses the `class` attribute to identify groups of elements that share a similar characteristic or property. The following `h1` element and paragraph element both belong to the `intro` class of elements:

```
<h1 class="intro">Tri and Succeed Sports</h1>
<p class="intro"> ... </p>
```

To select an element based on its `class` value, use the selector

`elem.class`

where `class` is the value of the `class` attribute. Thus the following style rule displays the text of `h1` headings from the `intro` class in blue:

```
h1.intro {color: blue;}
```

### TIP

An element can belong to several classes by including the class names in a space-separated list in the `class` attribute.

To apply the same style rule to all elements of a particular class, omit the element name. The following style rule displays the text of all elements from the `intro` class in blue:

```
.intro {color: blue;}
```

While `id` and `class` are the most common attributes to use with selectors, any attribute or attribute value can be the basis for a selector. Figure 2–15 lists all of the CSS attribute selector patterns based on attributes and attribute values.

Figure 2–15

Attribute selectors

Selector	Selects	Example	Selects
<code>elem#id</code>	Element <code>elem</code> with the ID value <code>id</code>	<code>h1#intro</code>	The <code>h1</code> heading with the id <code>intro</code>
<code>#id</code>	Any element with the ID value <code>id</code>	<code>#intro</code>	Any element with the id <code>intro</code>
<code>elem.class</code>	All <code>elem</code> elements with the class attribute value <code>class</code>	<code>p.main</code>	All paragraphs belonging to the <code>main</code> class
<code>.class</code>	All elements with the class value <code>class</code>	<code>.main</code>	All elements belonging to the <code>main</code> class
<code>elem[att]</code>	All <code>elem</code> elements containing the <code>att</code> attribute	<code>a[href]</code>	All hypertext elements containing the <code>href</code> attribute
<code>elem[att="text"]</code>	All <code>elem</code> elements whose <code>att</code> attribute equals <code>text</code>	<code>a[href="top.html"]</code>	All hypertext elements whose <code>href</code> attribute equals <code>top.html</code>
<code>elem[att~="text"]</code>	All <code>elem</code> elements whose <code>att</code> attribute contains the word <code>text</code>	<code>a[rel~="glossary"]</code>	All hypertext elements whose <code>rel</code> attribute contains the word <code>glossary</code>
<code>elem[att = "text"]</code>	All <code>elem</code> elements whose <code>att</code> attribute value is a hyphen-separated list of words beginning with <code>text</code>	<code>p[id ="first"]</code>	All paragraphs whose <code>id</code> attribute starts with the word <code>first</code> in a hyphen-separated list of words
<code>elem[att^="text"]</code>	All <code>elem</code> elements whose <code>att</code> attribute begins with <code>text</code>	<code>a[rel^="prev"]</code>	All hypertext elements whose <code>rel</code> attribute begins with <code>prev</code>
<code>elem[att\$="text"]</code>	All <code>elem</code> elements whose <code>att</code> attribute ends with <code>text</code>	<code>a[href\$="org"]</code>	All hypertext elements whose <code>href</code> attribute ends with <code>org</code>
<code>elem[att*="text"]</code>	All <code>elem</code> elements whose <code>att</code> attribute contains the value <code>text</code>	<code>a[href*="faq"]</code>	All hypertext elements whose <code>href</code> attribute contains the text string <code>faq</code>

## REFERENCE

## Using Attribute Selectors

- To select an element based on its ID, use the `elem#id` or `#id` selector, where `elem` is the name of the element and `id` is the value of the `id` attribute.
- To select an element based on its class value, use the `.class` or the `elem.class` selectors, where `class` is the value of the `class` attribute.
- To select an element that contains an `att` attribute, use `elem[att]`.
- To select an element based on whether its attribute value equals a specified value, `val`, use `elem[att="val"]`.

In the Tri and Succeed Sports home page, the main content is enclosed within an `article` element with the ID `about_tss`. Alison wants the `h1` and `h2` heading styles you entered in the last session to be applied only to `h1` and `h2` elements within articles that have this particular ID. Revise the style sheet now.

### To apply an id selector:

- 1. Return to the **tss\_styles.css** file in your editor.
- 2. Change the selectors for the **h1** and **h2** elements in the Heading Styles section to **article#about\_tss h1** and **article#about\_tss h2** respectively.

Figure 2–16 highlights the revised selectors in the style sheet.

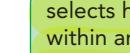
Figure 2–16

### Using an id selector

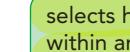
```
/* Heading Styles */
article#about_tss h1 {
 color: white;
 background-color: rgb(222, 128, 60);
}

article#about_tss h2 {
 color: white;
 background-color: rgb(235, 177, 131);
}
```

selects h1 headings within an article element with the about\_tss id



selects h2 headings within an article element with the about\_tss id



- 3. Save your changes to the file and then reload the **tss\_home.html** file in your browser. Verify that the design of the **h1** and **h2** headings is only applied to the headings in the **about\_tss** article but not to the other headings on the page.

The **article** element will be used in other pages in the Tri and Succeed Sports website. Alison has provided you with three additional HTML files containing descriptions of the services her company offers for runners, cyclists, and swimmers. On those pages the **article** elements have the **class** attribute with the value *syllabus*. Create style rules for the **h1** and **h2** elements within the articles on those pages.

### To apply a class selector:

- 1. Use your editor to open the **tss\_run\_txt.html**, **tss\_bike\_txt.html**, and **tss\_swim\_txt.html** files from the **html02 ▶ tutorial** folder. Enter **your name** and **the date** in the comment section of each file and save them as **tss\_run.html**, **tss\_bike.html**, and **tss\_swim.html** respectively.
- 2. Within each of the three files insert the following **link** elements directly before the closing **</head>** tag to link these files to the **tss\_layout.css** and **tss\_styles.css** files, respectively:
 

```
<link href="tss_layout.css" rel="stylesheet" />
<link href="tss_styles.css" rel="stylesheet" />
```
- 3. Take some time to study the content and structure of the files. Note that the **article** element has the **class** attribute with the value *syllabus*. Save your changes to the files.
- 4. Return to the **tss\_style.css** file in your editor.

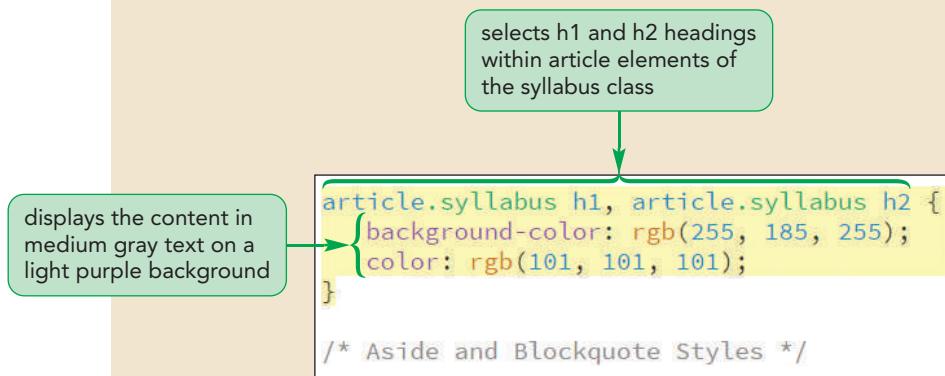
- 5. Within the Heading Styles section, add the following style rule to display the text of h1 and h2 headings in medium gray on a light purple background:

```
article.syllabus h1, article.syllabus h2 {
 background-color: rgb(255, 185, 255);
 color: rgb(101, 101, 101);
}
```

Figure 2–17 highlights the new style rule in the file.

Figure 2–17

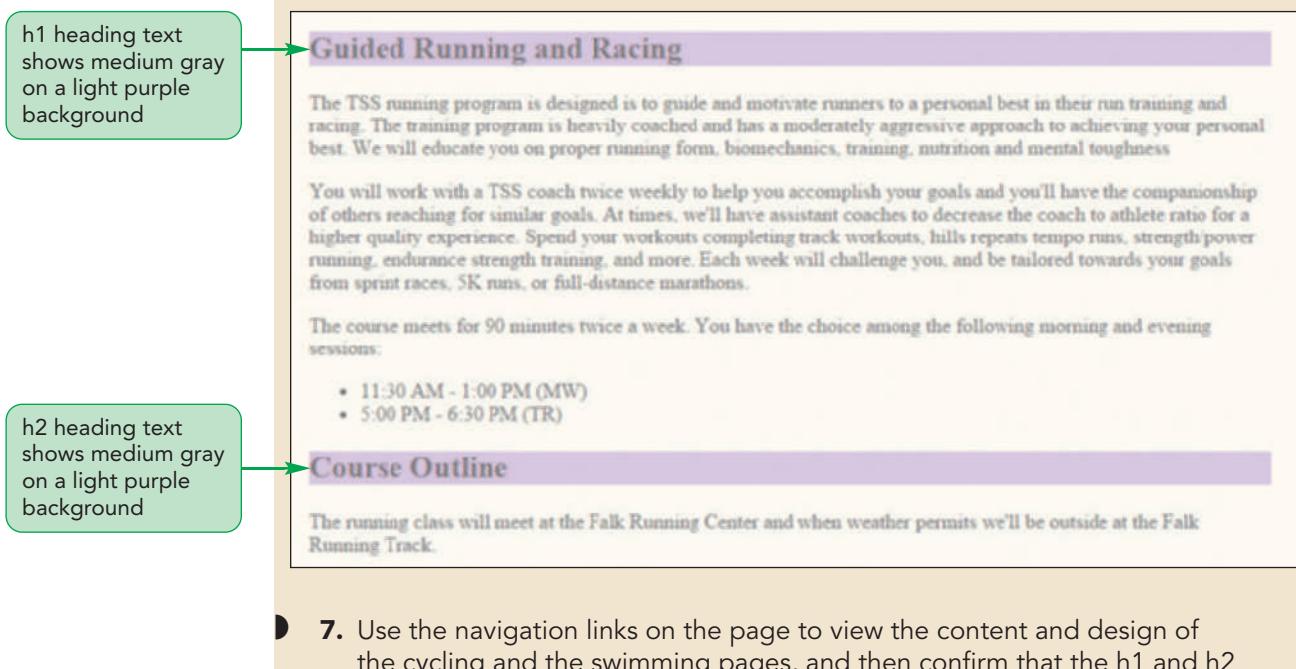
### Using a class selector



- 6. Save your changes to the style sheet and then open the **tss\_run.html** file in your browser. Figure 2–18 shows the appearance of the h1 and h2 headings on this page.

Figure 2–18

### Headings on the running class page



## INSIGHT

**Calculating Selector Specificity**

The general rule in CSS is that the more specific selector takes precedence over the more general selector, but the application of this rule is not always clear. For example, which of the following selectors is the more specific?

`header h1.top`

vs.

`#main h1`

To answer that question, CSS assigns a numeric value to the specificity of the selector using the formula

`(inline, ids, classes, elements)`

where *inline* is 1 for an inline style and 0 otherwise, *ids* is 1 for every id in the selector, *classes* is 1 for every class or attribute in the selector, and *elements* is 1 for every element in the selector. For example, the selector `ul#links li.first` would have a value of (0, 1, 1, 2) because it references one id value (`#links`), 1 class value (`.first`) and two elements (`ul` and `li`). Specificity values are read from left to right with a larger number considered more specific than a smaller number.

To answer our earlier question: the selector `header h1.top` has a value of (0, 0, 1, 2) but `#main h1` has a value of (0, 1, 0, 1) and, thus, is considered more specific because 0101 is larger than 0012.

By the way, since every inline style has the value (1, 0, 0, 0), they will always be more specific than any style set in an embedded or external style sheet.

**Working with Fonts**

**Typography** is the art of designing the appearance of characters and letters on a page. So far, the only typographic style you've used is the `color` property to set the text color. For the rest of this session, you'll explore other properties in the CSS family of typographical styles, starting with choosing the text font.

**Choosing a Font**

Text characters are based on **fonts** that define the style and appearance of each character in the alphabet. The default font used by most browsers for displaying text is Times New Roman, but you can specify a different font for any page element using the following `font-family` property

`font-family: fonts;`

where *fonts* is a comma-separated list, also known as a **font stack**, of specific or generic font names. A **specific font** is a font that is identified by name, such as Times New Roman or Helvetica, and based on a font definition file that is stored on the user's computer or accessible on the web. A **generic font** describes the general appearance of the characters in the text but does not specify any particular font definition file. Instead, the font definition file is selected by the browser to match the general characteristics of the generic font. CSS supports the following generic font groups:

- **serif**—a typeface in which a small ornamentation appears at the tail end of each character
- **sans-serif**—a typeface without any serif ornamentation
- **monospace**—a typeface in which each character has the same width; often used to display programming code

- **cursive**—a typeface that mimics handwriting with highly stylized elements and flourishes; best used in small doses for decorative page elements
- **fantasy**—a highly ornamental typeface used for page decoration; should never be used as body text

Because you have no control over which font definition file the browser will choose for a generic font, the common practice is to list specific fonts first, in order of preference, ending the font stack with a generic font. If the browser cannot find any of the specific fonts listed, it uses a generic font of its own choosing. For example, the style

```
font-family: 'Arial Black', Gadget, sans-serif;
```

tells a browser to use the Arial Black font if available; if not, to look for the Gadget font; and if neither of those fonts are available, to use its own generic sans-serif font. Note that font names containing one or more blank spaces (such as Arial Black) must be enclosed within single or double quotes.

Because the available fonts vary by operating system and device, the challenge is to choose a font stack limited to **web safe fonts**, which are fonts that will be displayed in mostly the same way in all operating systems and on all devices. Figure 2–19 lists several commonly used web safe font stacks.

Figure 2–19

Web safe font stacks

<b>Arial</b> abcdefghijklmnopqrstuvwxyz/1234567890 font-family: Arial, Helvetica, sans-serif;	<b>Lucida Console</b> abcdefghijklmnopqrstuvwxyz/1234567890 font-family: 'Lucida Console', Monaco, monospace;
<b>Arial Black</b> abcdefghijklmnopqrstuvwxyz/1234567890 font-family: 'Arial Black', Gadget, sans-serif;	<b>Lucida Sans Unicode</b> abcdefghijklmnopqrstuvwxyz/1234567890 font-family: 'Lucida Sans Unicode', 'Lucida Grande', sans-serif;
<b>Century Gothic</b> abcdefghijklmnopqrstuvwxyz/1234567890 font-family: 'Century Gothic', sans-serif;	<b>Palatino Linotype</b> abcdefghijklmnopqrstuvwxyz/1234567890 font-family: 'Palatino Linotype', 'Book Antiqua', Palatino, serif;
<b>Comic Sans MS</b> abcdefghijklmnopqrstuvwxyz/1234567890 font-family: 'Comic Sans MS', cursive;	<b>Tahoma</b> abcdefghijklmnopqrstuvwxyz/1234567890 font-family: Tahoma, Geneva, sans-serif;
<b>Courier New</b> abcdefghijklmnopqrstuvwxyz/1234567890 font-family: 'Courier New', Courier, monospace;	<b>Times New Roman</b> abcdefghijklmnopqrstuvwxyz/1234567890 font-family: 'Times New Roman', Times, serif;
<b>Georgia</b> abcdefghijklmnopqrstuvwxyz/1234567890 font-family: Georgia, serif;	<b>Trebuchet MS</b> abcdefghijklmnopqrstuvwxyz/1234567890 font-family: 'Trebuchet MS', Helvetica, sans-serif;
<b>Impact</b> abcdefghijklmnopqrstuvwxyz/1234567890 font-family: Impact, Charcoal, sans-serif;	<b>Verdana</b> abcdefghijklmnopqrstuvwxyz/1234567890 font-family: Verdana, Geneva, sans-serif;

**TIP**

Including too many fonts can make your page difficult to read. Don't use more than two or three typefaces within a single page.

A general rule for printing is to use sans-serif fonts for headlines and serif fonts for body text. For computer monitors, which have lower resolutions than printed material, the general rule is to use sans-serif fonts for headlines and body text, leaving serif fonts for special effects and large text.

Currently, the body text for the Tri and Succeed Sports website is based on a serif font selected by the browser. You'll add the following font stack for sans-serif fonts, which will take precedence over the browser font style rule:

```
font-family: Verdana, Geneva, sans-serif;
```

As a result of this style rule, the browser will first try to load the Verdana font, followed by the Geneva font. If both of these fonts are unavailable, the browser will load a generic sans-serif font of its own choosing. Add this font family to the style rule for the page body.

Font stacks should be listed in a comma-separated list with the most desired fonts listed first.

### To specify a font family for the page body:

- 1. Return to the **tss\_styles.css** file in your editor.
- 2. Add the following style to the style rule for the body element:

```
font-family: Verdana, Geneva, sans-serif;
```

Figure 2–20 highlights the new style for the body element.

Figure 2–20 Specifying a font stack

browser attempts to use the Verdana font first, followed by Geneva, and finally any generic sans-serif font

```
body {
 color: rgb(91, 91, 91);
 background-color: ivory;
 font-family: Verdana, Geneva, sans-serif;
}
```

- 3. Save your changes to the file and then reload the **tss\_home.html** file in your browser. Figure 2–21 shows the revised appearance of the body text using the sans-serif font.

Figure 2–21 Sans-serif font applied to the home page

<b>Links</b>	<b>About TSS</b>	<b>Comments</b>
<ul style="list-style-type: none"> <li>• <a href="#">Home</a></li> <li>• <a href="#">Running</a></li> <li>• <a href="#">Cycling</a></li> <li>• <a href="#">Swimming</a></li> <li>• <a href="#">Active.com</a></li> <li>• <a href="#">Runner's World</a></li> <li>• <a href="#">endomondo.com</a></li> <li>• <a href="#">Strava</a></li> <li>• <a href="#">Bicycling Magazine</a></li> <li>• <a href="#">VeloNews</a></li> <li>• <a href="#">Bicycle Tutor</a></li> <li>• <a href="#">Swim Smooth</a></li> <li>• <a href="#">Swimming World</a></li> <li>• <a href="#">USA Swimming</a></li> <li>• <a href="#">triathlon.org</a></li> <li>• <a href="#">usatriathlon.org</a></li> <li>• <a href="#">Texas Triathlons</a></li> <li>• <a href="#">CapTex Triathlon</a></li> <li>• <a href="#">Triathlon Calendar</a></li> <li>• <a href="#">Triathlete.com</a></li> <li>• <a href="#">Trifuel.com</a></li> </ul>	<p>Since 2002, <b>Tri and Succeed Sports</b> has provided Austin with a first class training center for athletes of all abilities and goals. We specialize in helping you reach your full potential. You tell us what you want to do; we work to fulfill your needs.</p> <p>Want to swim? Great! Interested in improving your cycling? Fantastic! Want to tackle a triathlon? We're there for you: before, during, and after the race. Or do you just want to get more fit? We are on it. We customize our instruction to match your goals. And you will finish what you start.</p> <p><b>Classes</b></p> <p>Winter instruction starts soon. Get a jump on your summer goals by joining us for individual or group instruction in:</p> 	<p>Thank you for all that you have done. I am amazed at my progress. I realize that I have lofty goals but you have me well on my way.</p> <p>Alison kept me focused working toward my dreams. She fosters a supportive and caring environment for growth as an athlete and as a person. Thank you!</p>

© Monkey Business Images/Shutterstock.com

- 4. View the other three pages in the website to verify that the sans-serif font is also applied to the body text on those pages.

## Exploring Web Fonts

Because web safe fonts limit your choices to a select number of fonts that have universal support, another approach is to supply a **web font** in which the definition font is supplied to the browser in an external file. Figure 2–22 describes the different web font file formats and their current levels of browser support. The format most universally accepted in almost all current browsers and on almost all devices is the Web Open Font Format (WOFF).

Figure 2-22 Web font formats

Format	Description	Browser
Embedded OpenType (EOT)	A compact form of OpenType fonts designed for use as embedded fonts in style sheets	Internet Explorer (IE)
TrueType (TTF)	Font standard used on the Mac OS and Microsoft Windows operating systems	IE, Firefox, Chrome, Safari, Opera
OpenType (OTF)	Font format built on the TrueType format developed by Microsoft	IE, Firefox, Chrome, Safari, Opera
Scalable Vector Graphics (SVG)	Font format based on an XML vocabulary designed to describe resizable graphics and vector images	Safari
Web Open Font Format (WOFF)	The W3C recommendation font format based on OpenType and TrueType with compression and additional metadata	IE, Firefox, Chrome, Safari, Opera

Web font files can be downloaded from several sites on the Internet. In many cases, you must pay for their use; in some cases, the fonts are free but are licensed only for non-commercial use. You should always check the EULA (End User License Agreement) before downloading and using a web font to make sure you are in compliance with the license. Finally, many web fonts are available through Web Font Service Bureaus that supply web fonts on their servers, which page designers can link to for a fee.

The great advantage of a web font is that it gives the author more control over the fonts used in the document; the disadvantage is that it becomes another file for the browser to download, adding to the time required to render the page. This can be a huge issue with mobile devices in which you want to limit the number and size of files downloaded by the browser.

## The @font-face Rule

To access and load a web font, you add the following `@font-face` rule to the style sheet

```

@font-face {
 font-family: name;
 src: url('url1') format('text1'),
 url('url2') format('text2'),
 ...
 descriptor1: value1;
 descriptor2: value2;
 ...
}

```

### TIP

It is considered best practice to always include a format value to alert the browser about the font's format so that it doesn't download a font definition file it can't display.

where `name` is the name of the font, `url` is the location of the font definition file, `text` is an optional text description of the font format, and the `descriptor: value` pairs are optional style properties that describe when the font should be used. Note several font definition files can be placed in a comma-separated list, allowing the browser to pick the file format it supports. For example, the following `@font-face` rule defines a font named Gentium installed from either the `Gentium.woff` file or if that fails, the `Gentium.ttf` file:

```

@font-face {
 font-family: Gentium;
 src: url('Gentium.woff') format('woff'),
 url('Gentium.ttf') format('truetype');
}

```

If the style sheet includes instructions to display a web font in italics, boldface, or other variants, the browser will modify the font, which sometimes results in poorly rendered text. However if the manufacturer has supplied its own version of the font variant, you can direct the browser to use that font file. For example, the following @font-face rule directs the browser to use the GentiumBold.woff or GentiumBold.ttf file when it needs to display Gentium in bold.

```
@font-face {
 font-family: Gentium;
 src: url('GentiumBold.woff') format('woff'),
 url('GentiumBold.ttf') format('truetype');
 font-weight: bold;
}
```

Note that the web font is given the same font-family name Gentium, which is the font name you use in a font stack. The added *descriptor: value* pair and *font-weight: bold* declarations tell the browser that these font files should be used with boldface Gentium.

Once you've defined a web font using the @font-face rule, you can include it in a font stack. For example, the following style will attempt to load the Gentium font first, followed by Arial Black, Gadget, and then a sans-serif font of the browser's choosing:

```
font-family: Gentium, 'Arial Black', Gadget, sans-serif;
```

Alison decides that the rendering of the Verdana font in the h1 and h2 heading text is too thick and heavy. She has located a web font named Quicksand that she is free to use under the End User License Agreement and she thinks it would work better for the page headings. She asks you to add this font to the style sheet and apply it to all h1 and h2 elements.

### TIP

The @font-face rule should be placed at the top of the style sheet after the @charset rule and before any styles that use the web font.

### To install and use a web font:

1. Return to the **tss\_styles.css** file in your editor.

2. Directly after the @charset rule at the top of the file, insert the following @font-face rule:

```
@font-face {
 font-family: Quicksand;
 src: url('Quicksand-Regular.woff') format('woff'),
 url('Quicksand-Regular.ttf') format('truetype');
}
```

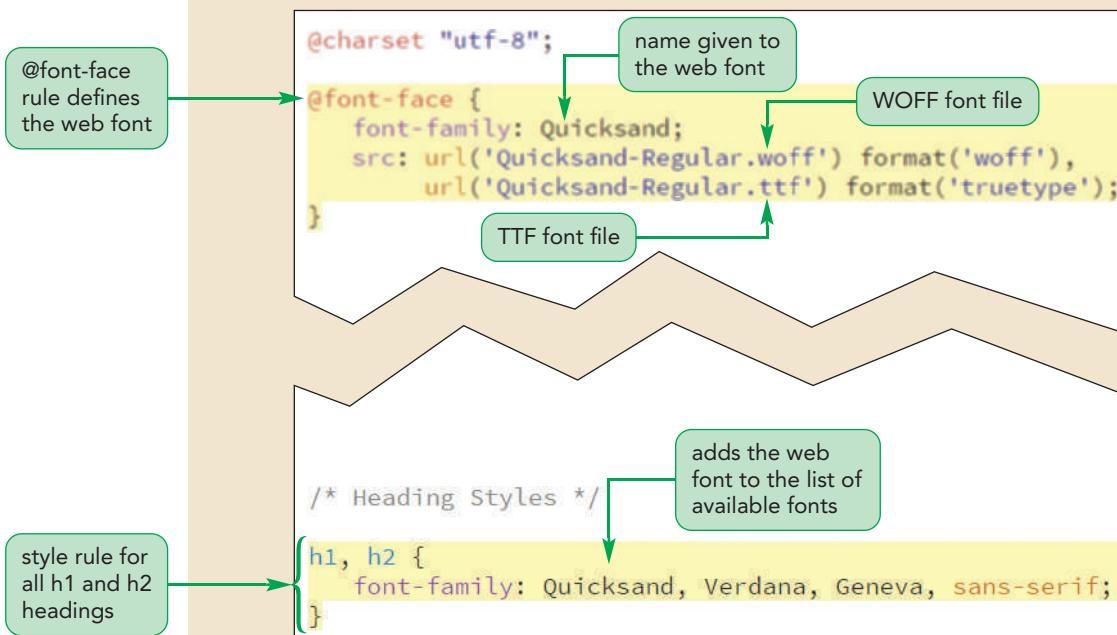
3. At the top of the Heading Styles section, insert the style rule:

```
h1, h2 {
 font-family: Quicksand, Verdana, Geneva, sans-serif;
}
```

Figure 2–23 highlights the code to create and use the Quicksand web font.

Figure 2-23

Accessing a web font



- 4. Save your changes to the file and reload the `tss_home.html` file in your browser. Figure 2-24 shows the revised appearance of the `h1` and `h2` headings using the Quicksand web font.

Figure 2-24

Quicksand font used for all h1 and h2 headings



© Ysbrand Cosijn/Shutterstock.com; © Charles T. Bennett/Shutterstock.com; © ostill/Shutterstock.com;  
© Monkey Business Images/Shutterstock.com

### INSIGHT Using Google Fonts

Google Fonts ([google.com/fonts](http://google.com/fonts)) hosts a library of free web fonts. After selecting a font from the Google Font catalog, you will receive the code for the `link` element to access the font files. For example, the following `link` element accesses a style sheet for a Google font named Monoton:

```
<link href="http://fonts.googleapis.com/css?family=Monoton"
 rel="stylesheet" /> → html
```

To use the Monoton font, include the following `font-family` property in the CSS style sheet:

```
font-family: Monoton, fantasy; → CSS
```

Google fonts, like all web fonts, need to be used in moderation because they can greatly increase the load times for your website. To help you know when you have exceeded a reasonable limit, the Google Fonts page shows a timer estimating the load times for all of the fonts you have selected. You can also limit the size of the font file by using the `&text` parameter to specify only those characters you want to download. For example, the following `link` element limits the Monoton font file to only the characters found in "TSS Sports":

```
<link href="http://fonts.googleapis.com/css?family=Monoton
 &text=TSS%20Sports" rel="stylesheet" />
```

Note that blank spaces are indicated using the `%20` character. If you have a longer text string, you can shorten the value of the `href` attribute by removing duplicate characters, as the order of characters doesn't matter.

## Setting the Font Size

Another important consideration in typography is the text size, which is defined using the following `font-size` property

```
font-size: size;
```

where `size` is a length in a CSS unit of measurement. Size values for any of these measurements can be whole numbers (0, 1, 2 ...) or decimals (0.5, 1.6, 3.9 ...). Lengths (and widths) in CSS are expressed in either absolute units or relative units.

### Absolute Units

**Absolute units** are units that are fixed in size regardless of the output device and are usually used only with printed media. They are specified in one of five standard units of measurement: `mm` (millimeters), `cm` (centimeters), `in` (inches), `pt` (points), and `pc` (picas). For example, to set the font size of your page body text to a 12pt font, you would apply the following style rule:

```
body {font-size: 12pt;}
```

Note that you should not insert a space between the size value and the unit abbreviation.

### Relative Units

Absolute units are of limited use because, in most cases, the page designer does not know the exact properties of the device rendering the page. In place of absolute units, designers use **relative units**, which are expressed relative to the size of other objects within the web page or relative to the display properties of the device itself.

The basic unit for most devices is the **pixel (px)**, which represents a single dot on the output device. A pixel is a relative unit because the actual pixel size depends on the resolution and density of the output device. A desktop monitor might have a pixel density of about 96ppi (pixels per inch), laptops are about 100 to 135ppi, while mobile phones have dense displays at 200 to 300ppi or more. Typically, most browsers will apply a base font size of 16px to body text with slightly larger font sizes applied to h1, h2, and h3 headings. You can override these default sizes with your own style sheet. The following style rules set the font size of the text on the page body to 10px and the font size of all h1 headings text to 14px:

```
body {font-size: 10px;}
h1 {font-size: 14px;}
```

#### TRY IT

You can explore typographic styles using the `demo_css.html` file in the `html02 ▶ demo` folder.

The exact appearance of the text depends greatly on the device's pixel density. While a 10px font might be fine on a desktop monitor, that same font size could be unreadable on a mobile device.

## Scaling Fonts with ems and rems

Because the page designer doesn't know the exact properties of the user's device, the common practice is to make the text **scalable** with all font sizes expressed relative to a default font size. There are three relative measurements used to provide scalability: percentages, ems, and rems.

A percentage sets the font size as a percent of the font size used by the containing element. For example, the following style rule sets the font size of an h1 heading to 200% or twice the font size of the h1 heading's parent element:

```
h1 {font-size: 200%;}
```

The em unit acts the same way as a percentage, expressing the font size relative to the font size of the parent element. Thus, to set the font size of h1 headings to twice the font size used in their parent elements, you can also use the style rule:

```
h1 {font-size: 2em;}
```

The em unit is the preferred style unit for web page text because it makes it easy to develop pages in which different page elements have consistent relative font sizes under any device.

Context is very important with relative units. For example, if the `h1` element is placed within a `body` element where the font size is 16px, the `h1` heading will have a font size twice that size or 32px. On the other hand, an `h1` heading nested within an `article` element where the font size is 9px will have a font size of 18px. In general, you can think of font sizes based on percentages and em units as relative to the size of immediately adjacent text.

The fact that relative units cascade through the style sheet can lead to confusing outcomes. For example, consider the following set of style rules for an `h1` element nested within an `article` element in the page body:

```
body {font-size: 16px;}
body > article {font-size: 0.75em;}
body > article > h1 {font-size: 1em;}
```

Glancing at the style rules, you might conclude that the font size of the `h1` element is larger than the font size used in the `article` element (since `1em > 0.75em`). However, this is not the case: both font sizes are the same. Remember, em unit expresses the text size relative to font size used in the parent element and since the `h1` heading is contained within the `article` element its font size of 1em indicates that it will have the same size used in the `article` element. In this case, the font size in the `article` element is 75% of 16px or 12 pixels as is the size of `h1` headings in the `article`.

Because of this confusion, some designers advocate using the **rem** or **root em unit** in which all font sizes are always expressed relative to the font size used in the `html`

element. Using rem, the following style rule sets the font size of article text to 75% of 16 pixels or 12 pixels while the h1 heading size is set to 16 pixels, equal to the font size of the html element:

```
html {font-size: 16px;}
article {font-size: 0.75rem;}
article > h1 {font-size: 1rem;}
```

The rem unit has become increasingly popular with designers as browser support grows and its use might possibly replace the use of the em unit as the font size unit of choice in upcoming years.

## Using Viewport Units

Another relative unit is the **viewport unit** in which lengths are expressed as a percentage of the width or height of the browser window. As the browser window is resized, the size of text based on a viewport unit changes to match. There are four viewport units: vw, vh, vmin, and vmax where

- 1vw = 1% of the browser window width
- 1vh = 1% of the browser window height
- 1vmin = 1vw or 1vh (whichever is smaller)
- 1vmax = 1vw or 1vh (whichever is larger)

For example, if the browser window is 1366 pixels wide, a length of 1vw would be equal to 13.66px. If the width of the window is reduced to 780 pixels, 1vw is automatically rescaled to 7.8 pixels. Auto-rescaling has the advantage that font sizes set with a viewport unit will be sized to match the browser window, maintaining a consistent page layout. The disadvantage is that page text can quickly become unreadable if the browser window becomes too small.

## Sizing Keywords

Finally, you also can express font sizes using the following keywords: xx-small, x-small, small, medium, large, x-large, xx-large, larger, or smaller. The font size corresponding to each of these keywords is determined by the browser. Note that the larger and smaller keywords are relative sizes, making the font size of the element one size larger or smaller than the font size of the container element. For example, the following style rules set the sidebar to be displayed in a small font, while an h1 element nested within that aside element is displayed in a font one size larger (medium):

```
aside {font-size: small;}
aside > h1 {font-size: larger;}
```

Use em units now to set the font size for the h1 and h2 headings, as well as the text within the navigation list and the aside element.

### To set font sizes of the page elements:

- 1. Return to the **tss\_styles.css** file in your editor.
- 2. Add the following style rules directly below the Heading Styles comment to define the font sizes for h1 and h2 headings throughout the website:

```
h1 {
 font-size: 2.2em;
}

h2 {
 font-size: 1.5em;
}
```

- 3. Go to the Aside and Blockquote Styles section and insert the following style rule to set the default font size of text in the aside element to 0.8em:

```
aside {
 font-size: 0.8em;
}
```

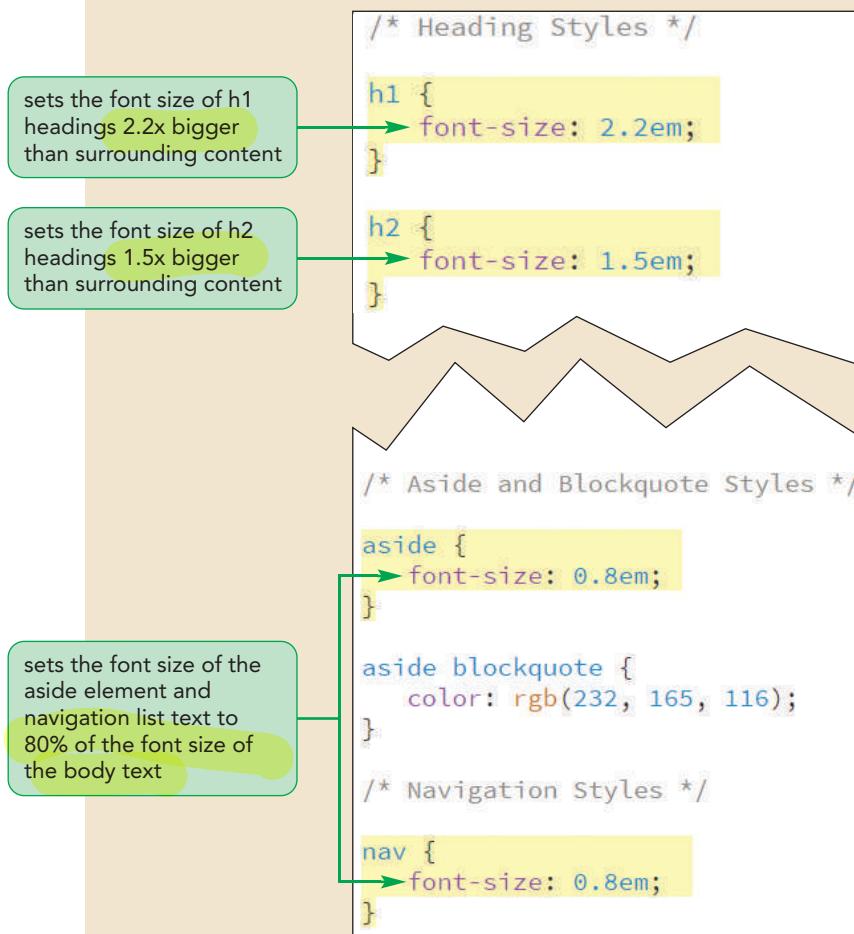
- 4. Go to the Navigation Styles section and insert the following style rule to set the default font size of text in the navigation list to 0.8em:

```
nav {
 font-size: 0.8em;
}
```

Figure 2–25 highlights the new font sizes for the website.

Figure 2–25

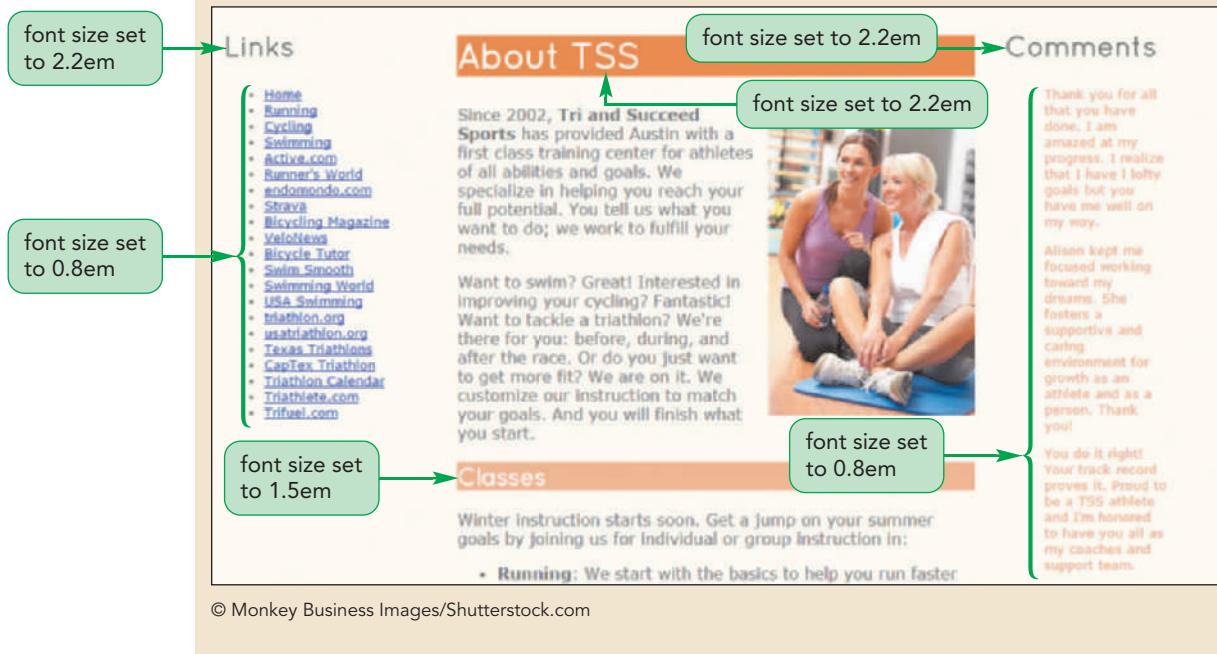
Setting font sizes for the website



5. Save your changes to the file and then reload the `tss_home.html` file in your browser. Figure 2–26 shows the revised font sizes of the headings, navigation list, and aside element.

Figure 2–26

Revised font sizes in the About TSS page



Note that the text of the `h1` heading in the page article is larger than the text in the `h1` headings from the navigation list and the aside element even though all headings have a font size of `2.2em`. This is because you reduced the default font size of the text in the navigation list and aside elements by 80% and thus the `h1` headings in those elements are also reduced by the same proportion.

## Controlling Spacing and Indentation

CSS supports styles to control some basic typographic attributes, such as kerning, tracking, and leading. Kerning sets the space between characters, while tracking sets the space between words. The properties to control an element's kerning and tracking are

`letter-spacing: value;`  
`word-spacing: value;`

where `value` is the size of space between individual letters or words. You specify these sizes with the same units that you use for font sizing. The default value for both kerning and tracking is 0 pixels. A positive value increases the letter and word spacing, while a negative value reduces the space between letters and words. If you choose to make your text scalable under a variety of devices and resolutions, you can express kerning and tracking values as percentages or em units.

**Leading** sets the space between lines of text and is defined with the following `line-height` property

`line-height: size;`

**TIP**

You can give multi-line titles more impact by tightening the space between the lines using a large font-size along with a small line-height.

where *size* is a value or a percentage of the font size of the text on the affected lines. If no unit is specified, the size value represents the ratio of the line height to the font size. The default value is 1.2 or 1.2em so that the line height is 20% larger than the font size. By contrast, the following style sets the line height to twice the font size, making the text appear double-spaced:

```
line-height: 2em;
```

An additional way to control text spacing is to set the indentation for the first line of a text block by using the following `text-indent` property

```
text-indent: size;
```

where *size* is expressed in absolute or relative units, or as a percentage of the width of the text block. For example, an indentation value of 5% indents the first line by 5% of the width of the block. The indentation value also can be negative, extending the first line to the left of the text block to create a **hanging indent**.

Alison suggests you increase the kerning used in the h1 and h2 headings to 0.1em so that the letters don't crowd each other on the page. She also asks that you increase the line height of the text of the navigation list to 2em so that the list of links on the home page is double-spaced.

**To set font sizes of the page elements:**

- 1. Return to the **tss\_styles.css** file in your editor.
- 2. In the Heading Styles section, insert the following style as part of the style rule for the h1, h2 selector:

```
letter-spacing: 0.1em;
```

- 3. Scroll down to the Navigation Styles section near the bottom of the file and add the following style rule for the text of ul elements nested within the nav element:

```
nav > ul {
 line-height: 2em;
}
```

Figure 2–27 highlights the letter-spacing and line-height styles for the website.

Figure 2–27

## Controlling letter spacing and line height

```

h1, h2 {
 font-family: Quicksand, Verdana, Geneva, sans-serif;
 letter-spacing: 0.1em; ← sets the space between
 letters to 0.1em
}

/* Navigation Styles */

nav {
 font-size: 0.8em;
}

nav > ul {
 line-height: 2em; ← double spaces the list
 of hypertext links
}

```

4. Save your changes to the file and then reload the `tss_home.html` file in your browser. Verify that the space between letters in the `h1` and `h2` headings has been increased and the list of links is now double-spaced.

By increasing the kerning in the headings, you've made the text appear less crowded, making it easier to read.

## Working with Font Styles

The style sheet built into your browser applies specific styles to key page elements; for instance, address elements are often displayed in italic, headings are often displayed in boldface. You can specify a different font style using the following `font-style` property

```
font-style: type;
```

where `type` is `normal`, `italic`, or `oblique`. The `italic` and `oblique` styles are similar in appearance, but might differ subtly depending on the font in use.

To change the weight of the text, use the following `font-weight` property

```
font-weight: weight;
```

where `weight` is the level of bold formatting applied to the text. CSS uses the keyword `bold` for boldfaced text and `normal` for non-boldfaced text. You also can use the keywords `bolder` or `lighter` to express the weight of the text relative to its surrounding content. Finally for precise weights, CSS supports weight values ranging from 100 (extremely light) up to 900 (extremely heavy) in increments of 100. In practice, however, it's difficult to distinguish font weights at that level of precision.

You can apply decorative features to text through the following `text-decoration` property

```
text-decoration: type;
```

where *type* equals `none` (for no decoration), `underline`, `overline`, or `line-through`. The `text-decoration` property supports multiple types so that the following style places a line under and over the element text:

```
text-decoration: underline overline;
```

Note that the `text-decoration` style has no effect on non-textual elements, such as inline images.

To control the case of the text within an element, use the following `text-transform` property

```
text-transform: type;
```

where *type* is `capitalize`, `uppercase`, `lowercase`, or `none` (to make no changes to the text case). For example, to capitalize the first letter of each word in an element, apply the style:

```
text-transform: capitalize;
```

Finally, CSS supports variations of the text using the `font-variant` property

```
font-variant: type;
```

where *type* is `normal` (for no variation) or `small-caps` (small capital letters). Small caps are often used in legal documents, such as software agreements, in which the capital letters indicate the importance of a phrase or point, but the text is made small so as not to detract from other elements in the document.

## Aligning Text Horizontally and Vertically

Text can be aligned horizontally or vertically within an element. To align the text horizontally, use the following `text-align` property

```
text-align: alignment;
```

where *alignment* is `left`, `right`, `center`, or `justify` (align the text with both the left and the right margins).

To vertically align the text within each line, use the `vertical-align` property

```
vertical-align: alignment;
```

where *alignment* is one of the keywords described in Figure 2–28.

**Figure 2–28** Values of the `vertical-align` property

Value	Description
<code>baseline</code>	Aligns the baseline of the element with the baseline of the parent element
<code>bottom</code>	Aligns the bottom of the element with the bottom of the lowest element in the line
<code>middle</code>	Aligns the middle of the element with the middle of the surrounding content in the line
<code>sub</code>	Subscripts the element
<code>super</code>	Superscripts the element
<code>text-bottom</code>	Aligns the bottom of the element with the bottom of the text in the line
<code>text-top</code>	Aligns the top of the element with the top of the text in the line
<code>top</code>	Aligns the top of the element with the top of the tallest object in the line

**TIP**

The subscript and superscript styles lower or raise text vertically, but do not resize it. To create true subscripts and superscripts, you also must reduce the font size.

Instead of using keywords, you can specify a length or a percentage for an element to be vertically aligned relative to the surrounding content. A positive value moves the element up as in the following style that raises the element by half the line height of the surrounding content:

```
vertical-align: 50%;
```

A negative value drops the content. For example, the following style drops the element an entire line height below the baseline of the current line:

```
vertical-align: -100%;
```

## Combining All Text Formatting in a Single Style

You can combine most of the text and font style properties into the following shorthand `font` property

```
font: style variant weight size/height family;
```

where `style` is the font's style, `variant` is the font variant, `weight` is the font weight, `size` is the font size, `height` is the height of each line, and `family` is the font stack. For example, the following style rule displays the element text in italic, bold, and small capital letters using Arial or another sans-serif font, with a font size of 1.5em and a line height of 2em:

```
font: italic small-caps bold 1.5em/2em Arial, sans-serif;
```

You do not have to include all of the values in the shorthand `font` property; the only required values are the `size` and `family` values. A browser assumes the default value for any omitted property; however, you must place any properties that you do include in the order indicated above.

At the bottom of each page in the Tri and Succeed Sports website, Alison has nested an `address` element within the body footer. The default browser style sheet displays address text in italics. Alison suggests that you display the text in a semi-transparent bold white font on a dark orange background and centered on the page. She also suggests that you use the small-cap font variant to add visual interest, and she wants you to increase the height of the address line to 3em. To make your CSS code more compact, you'll set all of the font values using the shorthand `font` property.

### To apply the `font` property:

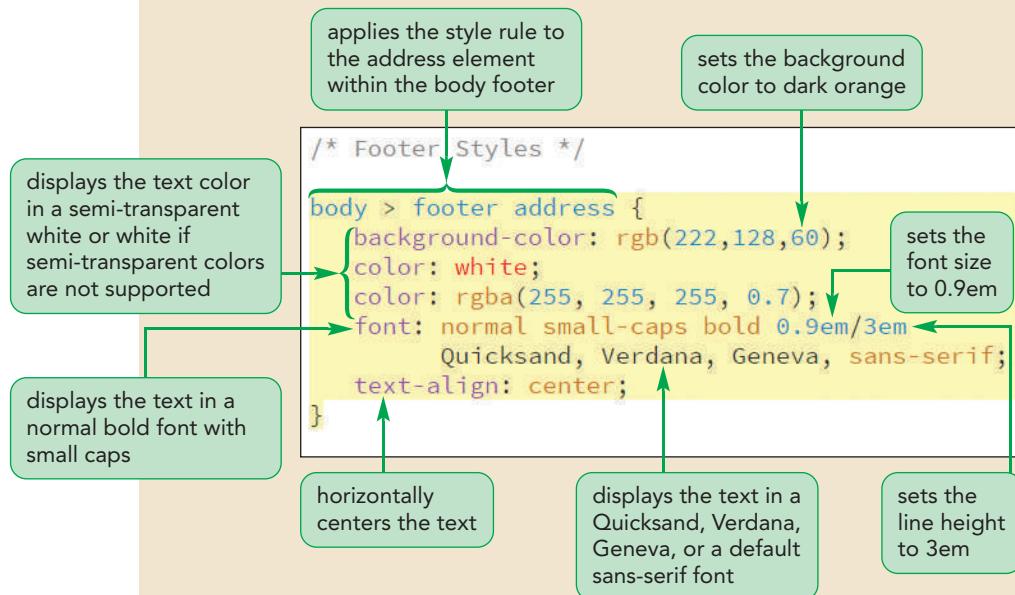
- 1. Return to the `tss_styles.css` file in your editor.
- 2. Go down to the Footer Styles section and add the following style rule:

```
body > footer address {
 background-color: rgb(222,128,60);
 color: white; color: rgba(255, 255, 255, 0.7);
 font: normal small-caps bold 0.9em/3em
 Quicksand, Verdana, Geneva, sans-serif;
 text-align: center;
}
```

Note that this style rule uses progressive enhancement by placing each color rule on its own line so that browsers that do not support semi-transparent colors will display the address text in white. Figure 2–29 highlights the style rule for the footer.

Figure 2-29

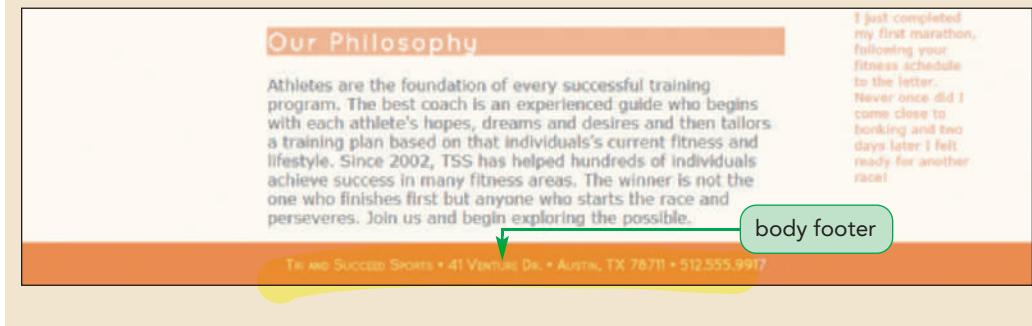
Style rule for the body footer



- 3. Save your changes to the file and then reload the tss\_home.html file in your browser. Figure 2-30 shows the revised appearance of the body footer.

Figure 2-30

Formatted body footer





## Decision Making: Selecting a Font

HTML and CSS provide several typographic options. Your main goal, however, is always to make your text easily readable. When designing your page, keep in mind the following principles:

- *Keep it plain*—Avoid large blocks of italicized text and boldfaced text. Those styles are designed for emphasis, not readability.
- *Sans-serif vs. serif*—Sans-serif fonts are more readable on a computer monitor and should be used for body text. Reserve the use of serif, cursive, and fantasy fonts for page headings and special decorative elements.
- *Relative vs. absolute*—Font sizes can be expressed in relative or absolute units. A relative unit like the em unit is more flexible and will be sized to match the screen resolution of the user's device, but you have more control over your page's appearance with an absolute unit. Generally, you want to use an absolute unit only when you know the configuration of the device the reader is using to view your page.
- *Size matters*—Almost all fonts are readable at a size of 14 pixels or greater; however, for smaller sizes, you should choose fonts that were designed for screen display, such as Verdana and Georgia. If you have to go really small (at a size of only a few pixels), you should either use a web font that is specially designed for that purpose or replace the text with an inline image.
- *Avoid long lines*—In general, try to keep the length of your lines to 60 characters or fewer. Anything longer is difficult to read.

When choosing any typeface and font style, the key is to test your selection on a variety of browsers, devices, screen resolutions, and densities. Don't assume that text that is readable and pleasing to the eye on your device screen will work as well on another device.

Alison likes the typographic changes you made to her website. In the next session, you'll explore how to design styles for hypertext links and lists, and you'll learn how to use CSS to add special visual effects to your web pages.

### Session 2.2 Quick Check

1. Which of the following selectors is used to match only paragraphs that are children of the `aside` element?
  - `aside p`
  - `p`
  - `aside > p`
  - All of the above
2. Which of the following selectors matches an `h1` heading with the id “`topHeading`”?
  - `h1#topHeading`
  - `h1[id="topHeading"]`
  - `h1`
  - All of the above

3. For the following style rules, what is the font size of the h1 heading in pixels?

```
body {font-size: 16px;}
body > article {font-size: 0.75em;}
body > article > h1 {font-size: 1.5em;}
a. 16px;
b. 12px;
c. 18px;
d. 18.25px;
```

16.  
0.75em.

2<sup>1</sup>  
28

4. Which of the following styles will set the font size to 2% of the viewport width?

- a. 2%vw;
- b. 2vw;
- c. 2vmin;
- d. 2%view;

5. Provide a style rule to remove underlining from hypertext links marked with the

<a> tag and nested within a navigation list.

- a. nav > a {text-decoration: none;}
- b. nav > a {text-decoration: no-underline;}
- c. nav a {text-decoration: none;}
- d. nav a {text-decoration: underline="no";}

<nav>  
<ul>  
<li> <a href="#"></a>  
: descendant

6. Provide an @font-face rule to create a web font named Cantarell based on the font file cantarell.woff and cantarell.ttf.

- a. @font-face {
 font-family: Cantarell;
 src: url('Cantarell.woff') format('woff'),
 src: url('Cantarell.ttf') format('truetype');
 }
- b. font-family: Cantarell, cantarell.woff, cantarell.ttf.
- c. Cantarell {
 src: url('Cantarell.woff') format('woff'),
 src: url('Cantarell.ttf') format('truetype');
 }
- d. All of the above will work

7. Which of the following provides a style rule to display all blockquote elements belonging to the Reviews class in italic and indented 3em?

- a. Reviews > blockquote {
 font-style: italic;
 indent: 3em;
 }
- b. blockquote#Reviews {
 font-style: italic;
 indent: 3em;
 }
- c. blockquote.Reviews {
 font-style: italic;
 text-indent: 3em;
 }
- d. blockquote#Reviews {
 italic: true;
 indent: 3em;
 }

8. Which of the following style rules will center the text of all h1 through h6 headings with the text displayed at normal weight?

- a. 

```
h1 - h6 {
 text-align: center;
 weight: normal;
}
```
- b. 

```
h1, h2, h3, h4, h5, h6 {
 align: center;
 weight: normal;
}
```
- c.**

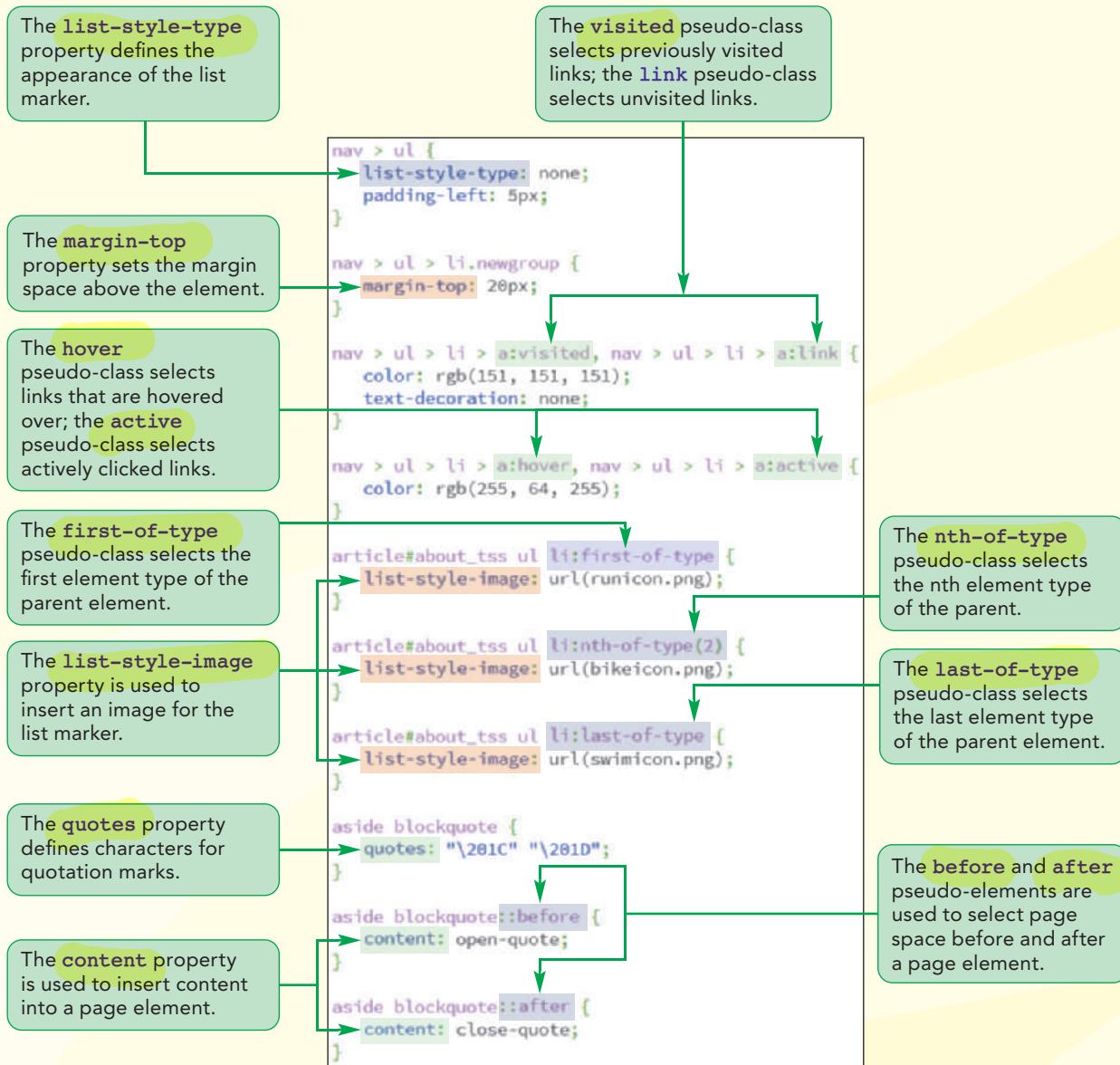
```
h1, h2, h3, h4, h5, h6 {
 text-align: center;
 font-weight: normal;
}
```
- d. 

```
h1 - h6 {
 text-align: center;
 font-weight: normal;
}
```

9. What style property sets the kerning of the text within an element?

- a.** letter-spacing
- b. word-spacing
- c. kerning-size
- d. kerning

## Session 2.3 Visual Overview:



© Monkey Business Images/Shutterstock.com;  
 © Courtesy Patrick Carey

# Pseudo Elements and Classes

Style of the link changes when the mouse pointer hovers over it.

Links

Home  
Running  
Cycling  
**Swimming**   
Active.com  
Runner's World  
endomondo.com  
Strava  
Bicycling Magazine  
VeloNews  
Bicycle Tutor  
Swim Smooth  
Swimming World  
USA Swimming

triathlon.org  
usatriathlon.org  
Texas Triathlons  
CapTex Triathlon  
Triathlon Calendar  
Triathlete.com  
Trifuel.com

Top margins at each newgroup class are set to 20 pixels.

## About TSS

Since 2002, **Tri and Succeed Sports** has provided Austin with a first class training center for athletes of all abilities and goals. We specialize in helping you reach your full potential. You tell us what you want to do; we work to fulfill your needs.

Want to swim? Great! Interested in improving your cycling? Fantastic! Want to tackle a triathlon? We're there for you: before, during, and after the race. Or do you just want to get more fit? We are on it. We customize our instruction to match your goals. And you will finish what you start.



Monkey Business Images/Shutterstock.com

## Comments

“Thank you for all that you have done. I am amazed at my progress. I realize that I have lofty goals but you have me well on my way.”

“Alison kept me focused working toward my dreams. She fosters a supportive and caring environment for growth as an athlete and as a person. Thank you!”

“You do it right! Your track record proves it. Proud to be a TSS athlete and I’m honored to have you all as my coaches and support team.”

“The coaches at TSS treat you with the highest respect: whether you’re an individual getting off the couch for the first time or an elite athlete training for the Iron Man. They know their stuff.”

“I just completed my first marathon, following your fitness schedule to the letter. Never once did I come close to bonking and two days later I felt ready for another race!”

## Classes

Winter instruction starts soon. Get a jump on your summer goals by joining us for individual or group instruction in:

 **Running:** We start with the basics to help you run faster and farther than you ever thought possible without aches and pains.

 **Cycling:** The indoor bike trainers at TSS include everything you need to refine your technique, stamina, and power for improved results on the road.

 **Swimming:** The open water swim can be one of the most frightening sports to master. Our classes begin with basic techniques so that your swim can be very enjoyable, and not a chore.

This is the first li element.

This is the last li element.

This is the second li element.

An image is used to mark each of the three list markers.

## Formatting Lists

In this session, you'll explore how to use CSS to create styles for different types of lists that you learned about in Tutorial 1. You'll start by examining how to create styles for the list marker.

### Choosing a List Style Type

The default browser style for unordered and ordered lists is to display each list item alongside a symbol known as a **list marker**. By default, unordered lists are displayed with a solid disc while ordered lists are displayed with numerals. To change the type of list marker or to prevent any display of a list marker, apply the following `list-style-type` property

```
list-style-type: type;
```

where `type` is one of the markers described in Figure 2–31.

Figure 2–31 Values of the `list-style-type` property

<code>list-style-type</code>	Marker(s)
<code>disc</code>	●
<code>circle</code>	○
<code>square</code>	■
<code>decimal</code>	1, 2, 3, 4, ...
<code>decimal-leading-zero</code>	01, 02, 03, 04, ...
<code>lower-roman</code>	i, ii, iii, iv, ...
<code>upper-roman</code>	I, II, III, IV, ...
<code>lower-alpha</code>	a, b, c, d, ...
<code>upper-alpha</code>	A, B, C, D, ...
<code>lower-greek</code>	α, β, γ, δ, ...
<code>upper-greek</code>	Α, Β, Γ, Δ, ...
<code>none</code>	no marker displayed

#### TIP

List style properties can be applied to individual list items using the `li` selector in the style rule.

For example, the following style rule marks each item from an ordered list with an uppercase Roman numeral:

```
ol {list-style-type: upper-roman;}
```

### Creating an Outline Style

Nested lists can be displayed in an outline style through the use of contextual selectors. For example, the following style rules create an outline style for a nested ordered list:

```
ol {list-style-type: upper-roman;}
ol ol {list-style-type: upper-alpha;}
ol ol ol {list-style-type: decimal;}
```

In this style, the `ol` selector selects the top level of the list, displaying the list items with a Roman numeral. The `ol ol` selector selects the second level, marking the items with capital letters. The third level indicated by the `ol ol ol` selector is marked with decimal values.

To see how these style rules are rendered on a page, you'll apply them to the three pages that Alison has set up describing the running, cycling, and swimming programs offered by Tri and Succeed sports. Each page contains a syllabus outlining the course of study for the next several weeks.

### To apply an outline style:

- 1. If you took a break after the previous session, make sure the **tss\_styles.css** file is open in your editor.
- 2. Scroll down to the List Styles section and insert the following style rules to format nested ordered lists within the syllabus article:

```
article.syllabus ol {
 list-style-type: upper-roman;
}

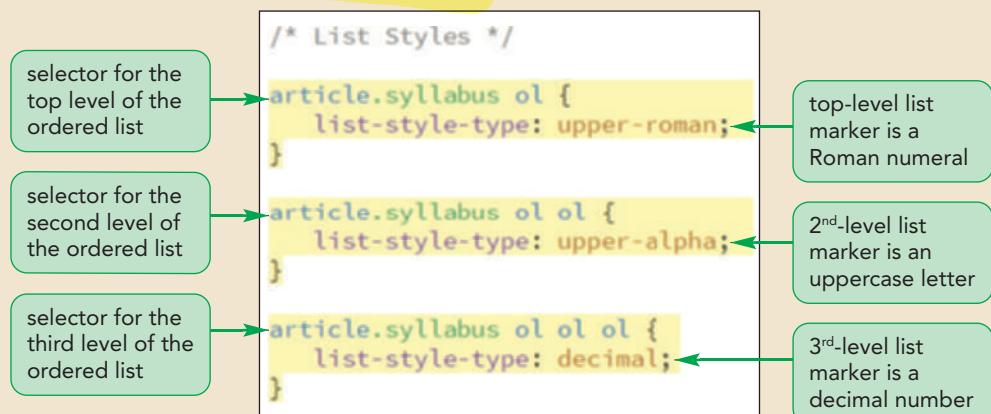
article.syllabus ol ol {
 list-style-type: upper-alpha;
}

article.syllabus ol ol ol {
 list-style-type: decimal;
}
```

Figure 2–32 highlights the style rule for the nested lists.

Figure 2–32

### Creating an outline style for a nested list



- 3. Save your changes to the file and then open the **tss\_run.html** file in your browser. As shown in Figure 2–33, the syllabus for the class should now be displayed in an outline style.

Figure 2-33

## Course outline

## Course Outline

The running class will meet at the Falk Running Center and, when weather permits, we'll be outside at the Falk Running Track.

- I. Week 1:
  - A. Orientation
    - 1. Setting a Goal
    - 2. Group Running
    - 3. Clothing and Shoes
    - 4. Danger Zones
  - B. Initial Assessment
    - 1. Gait Assessment
    - 2. Power Measure
    - 3. Time Trial
  - C. Stretching Techniques
- II. Week 2:
  - A. Wind Sprints
  - B. Recovery
  - C. Building your Core
- III. Week 3:
  - A. Wind Sprints 2
  - B. Stretching Session
  - C. Yoga and Running

nested list with different markers for each level of list items

Alison points out that the hypertext links from the navigation list are displayed with a disc marker. She asks you to remove the markers from the navigation list by setting the `list-style-type` property to `none`.

## To remove the markers from navigation lists:

- 1. Return to the `tss_styles.css` file in your editor.
- 2. Go to the Navigation Styles section and, within the style rule for the `nav > ul` selector, add the style `list-style-type: none;`

Figure 2-34 highlights the new style.

Figure 2-34

## Removing list markers from navigation lists

```
nav > ul {
 line-height: 2em;
 list-style-type: none;
}
```

displays no markers for unordered lists within the `nav` element

- 3. Save your changes to the file and then open the `tss_home.html` file in your browser. Verify that there are no markers next to the navigation list items in the left column.
- 4. Go to the other three pages in the website and verify that navigation lists in these pages also do not have list markers.

### Designing a List

- To define the appearance of the list marker, use the property

```
list-style-type: type;
```

where *type* is disc, circle, square, decimal, decimal-leading-zero, lower-roman, upper-roman, lower-alpha, upper-alpha, lower-greek, upper-greek, or none.

- To insert a graphic image as a list marker, use the property

```
list-style-image: url(url);
```

where *url* is the URL of the graphic image file.

- To set the position of list markers, use the property

```
list-style-position: position;
```

where *position* is inside or outside.

- To define all of the list style properties in a single style, use the property

```
list-style: type url(url) position;
```

## Using Images for List Markers

You can supply your own graphic image for the list marker with the following *list-style-image* property

```
list-style-image: url(url);
```

where *url* is the URL of a graphic file containing the marker image. Marker images are only used with unordered lists in which the list marker is the same for every list item. For example, the following style rule displays items from unordered lists marked with the graphic image in the *redball.png* file:

```
ul {list-style-image: url(redball.png);}
```

Alison has an icon image in a file named *runicon.png* that she wants to use for the classes listed on the Tri and Succeed Sports home page in the About TSS article. Apply her image file to the list now.

### To use an image for a list marker:

- 1. Return to the **tss\_styles.css** file in your editor.
- 2. At the top of the List Styles section, insert the following style rule:

```
article#about_tss ul {
 list-style-image: url(runicon.png);
}
```

Figure 2-35 highlights the style rule to use the *runicon.png* file as the list marker image.

Figure 2-35

## Displaying an image in place of a list marker

style rule applied to the unordered list within the about\_tss article

```
/* List Styles */
article#about_tss ul {
 list-style-image: url(runicon.png);
}
```

displays the runicon.png file as the list marker

- 3. Save your changes to the file and then open the **tss\_home.html** file in your browser. As shown in Figure 2-36 the items in the unordered list now use the runicon.png image file as their list marker.

Figure 2-36

## Unordered list with the runicon.png image marker

runicon.png image file

## Classes

Winter instruction starts soon. Get a jump on your summer goals by joining us for individual or group instruction in:

-  **Running:** We start with the basics to help you run faster and farther than you ever thought possible without aches and pains.
-  **Cycling:** The indoor bike trainers at TSS include everything you need to refine your technique, stamina, and power for improved results on the road.
-  **Swimming:** The open water swim can be one of the most frightening sports to master. Our classes begin with basic techniques so that your swim can be very enjoyable, and not a chore.

© Courtesy Patrick Carey

Notice that the list marker is aligned with the baseline of the first line in each list item. This is the default placement for list marker images.

## Setting the List Marker Position

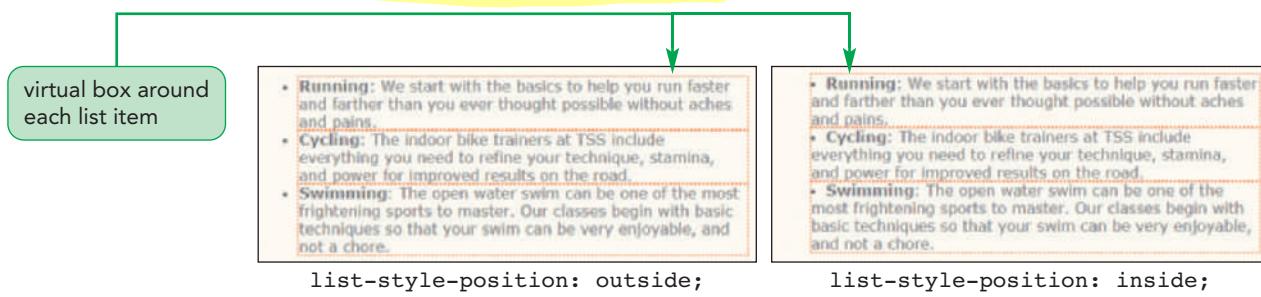
CSS treats each list item as a block-level element, placed within a virtual box in which the list marker is placed outside of the list text. You can change this default behavior using the following `list-style-position` property

```
list-style-position: position;
```

where `position` is either `outside` (the default) or `inside`. Placing the marker inside the virtual box causes the list text to flow around the marker. Figure 2-37 shows how the `list-style-position` property affects the flow of the text around the bullet marker.

Figure 2–37

## Values of the list-style-position property



All three of the list styles just discussed can be combined within the following shorthand `list-style` property:

`list-style: type image position;`

where `type` is the marker type, `image` is an image to be displayed in place of the marker, and `position` is the location of the marker. For example, the following style rule displays unordered lists using the marker found in the `bullet.png` image placed inside the containing block:

`ul {list-style: circle url(bullet.png) inside;}`

If a browser is unable to display the `bullet.png` image, it uses a default circle marker instead. You do not need to include all three style properties with the list style. Browsers will set any property you omit to the default value.

Allison notes that there is a lot of unused space to the left of the items in the navigation list now that the list markers have been removed. She wants you to move the navigation list into that empty space. To do this, you'll work with the CSS styles for margin and padding space.

## Working with Margins and Padding

Block-level elements like paragraphs or headings or lists follow the structure of the **box model** in which the content is enclosed within the following series of nested boxes:

- the content of the element itself
- the **padding space**, which extends from the element's content to the element's border
- the **border** surrounding the padding space and marking the extent of the element
- the **margin space** comprised of the space between the element and the next page element

Figure 2–38 shows a diagram of the box model for a sample paragraph discussing athletes at Tri and Succeed Sports.

Figure 2-38

The CSS box model

**TIP**

Your browser's developer tools will display a schematic diagram of the box model for each element on your page so that you can determine the size of the padding, border, and margin spaces.

The browser's internal style sheet sets the size of the padding, border, and margin spaces but you can specify different sizes in your style sheet.

## Setting the Padding Space

To set the width of the padding space, use the following `padding` property

`padding: size;`

where `size` is expressed in one of the CSS units of length or the keyword `auto` to let the browser automatically choose the padding. For example, the following style rule sets the padding space around every paragraph to 20 pixels:

`p {padding: 20px;}`

The padding space can also be defined for each of the four sides of the virtual box by writing the `padding` property as follows

`padding: top right bottom left;`

where `top` is the size of the padding space along the top edge of the content, `right` is padding along the right edge, `bottom` is the size of the bottom padding, and `left` is the size of the padding along the left edge. Thus, the following style rule creates a padding space that is 10 pixels on top, 0 pixels to the right, 15 pixels on the bottom, and 5 pixels to the left:

`p {padding: 10px 0px 15px 5px;}`

To help remember this order, think of moving clockwise around the box, starting with the top edge. While you don't have to supply values for all of the edges, the values you supply are interpreted based on how many values you supply. So, if you specify a single value, it's applied to all four sides equally. Two values set the padding spaces for the top/bottom edges and the right/left edges. For example, the following style rule sets the top and bottom padding spaces at 10 pixels and the right and left padding spaces at 5 pixels:

`p {padding: 10px 5px;}`

With three values, the padding spaces are set for the top, right/left, and bottom edges. Thus, the following rule sets the size of the top padding space to 10 pixels, the left/right spaces to 5 pixels, and the bottom space to 0 pixels:

`p {padding: 10px 5px 0px;}`

To define the padding space for one edge but not for the others, apply the following style properties:

 padding-top: *size*;  
 padding-right: *size*;  
 padding-bottom: *size*;  
 padding-left: *size*;

The following style rule sets the top padding of every paragraph to 10 pixels but it does not specify a padding size for any of the three remaining edges:

```
p {padding-top: 10px;}
```

With ordered and unordered lists, the default browser style sets the left padding space to 40 pixels in order to provide the extra space needed for the list markers. Removing the list markers doesn't remove this padding space. Allison suggests you recover this unused space by reducing the size of the left padding space in the navigation list to 5 pixels.

Include the unit in any style involving padding or margin spaces.

### To change the left padding used in the navigation list:

- 1. Return to the **tss\_styles.css** file in your editor.
- 2. Locate the `nav > ul` style rule in the Navigation Styles section and add the style `padding-left: 5px;`.

Figure 2-39 highlights the new style for all navigation lists.

Figure 2-39

### Setting the size of the left padding space

selects unordered lists within the nav element

```
nav > ul {
 line-height: 2em;
 list-style-type: none;
 padding-left: 5px;
}
```

sets the padding on the left edge to 5 pixels

- 3. Save your changes to the file and then reload the `tss_home.html` file in your browser. Verify that the entries in the navigation list in the left column have been shifted to the left, which is the result of changing the left padding setting to 5 pixels.

Now that you've worked with the padding space, you'll examine how to work with margins.

### Setting Padding and Margin Space

- To set the padding space around all sides of the element, use

```
padding: size;
```

where *size* is the size of the padding using one of the CSS units of length.

- To set the margin space around all sides of the element, use

```
margin: size;
```

- To set padding or margin on only one side (top, right, bottom, or left) include the name of the side in the property as

```
padding-side: size;
margin-side: size;
```

where *side* is *top*, *right*, *bottom*, or *left*.

- To set different padding or margins on each side of the element, enter the sides as

```
padding: top right bottom left;
margin: top right bottom left;
```

where *top*, *right*, *bottom*, and *left* are individual sizes for the associated side.

### Setting the Margin and the Border Spaces

Styles to set the margin space are similar to the padding styles. To set the size of the margin around your block elements, use either of the following properties:

```
margin: size;
```

or

```
margin: top right bottom left;
```

The margins of individual sides are set using the style properties

```
margin-top: size;
margin-right: size;
margin-bottom: size;
margin-left: size;
```

where once again *size* is expressed in one of the CSS units of length or using the keyword *auto* to have the browser automatically set the margin.

The size of the border space is set using the following *border-width* property

```
border-width: size;
```

or

```
border-width: top right bottom left;
```

or with the properties *border-top-width*, *border-right-width*, *border-bottom-width*, and *border-left-width* used to specify the size of individual borders. You'll explore borders in more detail in Tutorial 4.

The navigation list that Alison created for the home page groups the list into those links for pages within the Tri and Succeed Sports website and those links to external websites. The list item at the start of each group is marked with the *class* value *newgroup*. Alison suggests you increase the top margin above each group of links to 20 pixels in order to offset it from the preceding group. The groups will be easier to recognize after the top margin for each group has been increased.

### To increase the top margin:

- 1. Return to the **tss\_styles.css** file in your editor.
- 2. Directly below the style rule for the `nav > ul` selector in the Navigation Styles section, add the following rule:

```
nav > ul > li.newgroup {
 margin-top: 20px;
}
```

Figure 2–40 highlights the style rule setting the top margin value.

Figure 2–40

### Setting the size of the top margin

selects the list items belonging to the newgroup class found within the unordered navigation list

```
nav > ul {
 line-height: 2em;
 list-style-type: none;
 padding-left: 5px;
}

nav > ul > li.newgroup {
 margin-top: 20px;
}
```

sets the margin space on the top edge to 20 pixels

- 3. Save your changes to the file and then reload the `tss_home.html` file in your browser. Verify that the entries in the navigation list are now split into three groups: the first group containing the links from the Tri and Succeed Sports website; the second group containing links to websites on running, cycling, and swimming; and the third group containing links to triathlon websites.

Alison has also noticed that the block quotes in the right column of the home page have unused space to the left, leaving less space for the customer quotes. The default browser style for the `blockquote` element offsets block quotes from the surrounding text by setting the left and right margins to 40 pixels. To adjust this spacing and to make the block quotes more readable, you'll reduce the left/right margins to 5 pixels. You'll also increase the top/bottom margins to 20 pixels to better separate one customer quote from another.

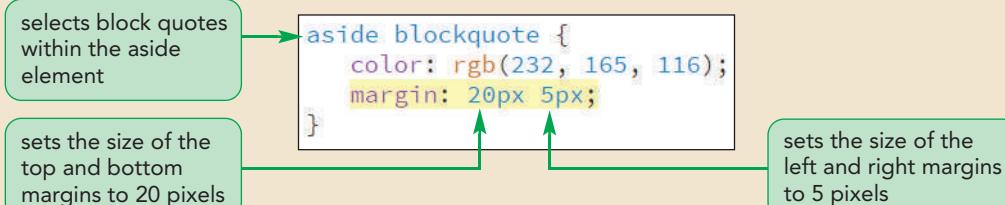
### To change the margin space around block quotes:

- 1. Return to the **tss\_styles.css** file in your editor.
- 2. Locate the style rule for the `aside blockquote` selector in the Aside and Blockquote Styles section and insert the `margin: 20px 5px;` style into the style rule for the `aside blockquote` selector.

Figure 2–41 displays the style to change the margin space around the `blockquote` element.

Figure 2-41

## Setting the margin size for block quotes



- 3. Save your changes to the file and then reload the `tss_home.html` file in your browser. Figure 2-42 displays the revised appearance of the page with the new padding and margin sizes applied to the navigation list and the block quotes.

Figure 2-42

## Home page with new margins and padding

each block quote surrounded by a 20 pixel top/bottom margin and a 5 pixel left/right margin

left padding set to 5 pixels

each new group offset by a 20 pixel top margin

**Links**

- Home
- Running
- Cycling
- Swimming
- Active.com
- Runner's World
- endomondo.com
- Strava
- Bicycling Magazine
- VeloNews
- Bicycle Tutor
- Swim Smooth
- Swimming World
- USA Swimming
- triathlon.org
- usatriathlon.org
- Texas Triathlons
- CapTex Triathlon
- Triathlon Calendar
- Triathlete.com

**About TSS**

Since 2002, **Tri and Succeed Sports** has provided Austin with a first class training center for athletes of all abilities and goals. We specialize in helping you reach your full potential. You tell us what you want to do; we work to fulfill your needs.

Want to swim? Great! Interested in improving your cycling? Fantastic! Want to tackle a triathlon? We're there for you: before, during, and after the race. Or do you just want to get more fit? We are on it. We customize our instruction to match your goals. And you will finish what you start.

**Classes**

Winter instruction starts soon. Get a jump on your summer goals by joining us for individual or group instruction in:

**Running:** We start with the basics to help you run faster and farther than you ever thought possible without aches and pains.

**Comments**

Thank you for all that you have done. I am amazed at my progress. I realize that I have lofty goals but you have me well on my way.

Alison kept me focused working toward my dreams. She fosters a supportive and caring environment for growth as an athlete and as a person. Thank you!

You do it right! Your track record proves it. Proud to be a TSS athlete and I'm honored to have you all as my coaches and support team.

The coaches at TSS treat you with the highest respect; whether you're an individual getting off the couch for the first time or an elite athlete training for the Iron Man. They know their stuff.

I just completed my first marathon, following your fitness schedule to the letter. Never once did I come close to bonking and

© Monkey Business Images/Shutterstock.com; © Courtesy Patrick Carey

Alison thinks the revised appearance of the navigation list and the customer quotes is a big improvement. However, she doesn't like the underlining in the navigation list. She would like the underlining to appear only when the user hovers the mouse pointer over the link. She would also like a different list marker to appear next to each list item in the classes section. You can make these changes using pseudo-classes and pseudo-elements.

## Using Pseudo-Classes and Pseudo-Elements

Not everything that appears in the rendered page is marked up in the HTML file. For example, a paragraph has a first letter or a first line but those are not marked as distinct elements. Similarly, an element can be classified based on a particular property without having a `class` attribute. The initial entry from an ordered list has the property of being the first item, but no `class` attribute in the HTML file identifies it as such. These elements and `class` attributes that exist only within the rendered page but not within the HTML document are known as pseudo-elements and pseudo-classes. Despite not being part of the HTML document, you can still write style rules for them.

### Pseudo-Classes

A **pseudo-class** is a classification of an element based on its current status, position, or use in the document. The style rule for a pseudo-class uses the selector

```
element:pseudo-class
```

where `element` is an element from the document and `pseudo-class` is the name of a CSS pseudo-class. Pseudo-classes are organized into structural and dynamic classes. A **structural pseudo-class** classifies an element based on its location within the structure of the HTML document. Figure 2–43 lists the structural pseudo-classes supported in CSS.

Figure 2–43

Structural pseudo-classes

Pseudo-Class	Matches
<code>:root</code>	The top element in the document hierarchy (the <code>html</code> element)
<code>:empty</code>	An element with no content
<code>:only-child</code>	An element with no siblings
<code>:first-child</code>	The first child of the parent element
<code>:last-child</code>	The last child of the parent element
<code>:first-of-type</code>	The first descendant of the parent that matches the specified type
<code>:last-of-type</code>	The last descendant of the parent that matches the specified type
<code>:nth-of-type(n)</code>	The $n^{\text{th}}$ element of the parent of the specified type
<code>:nth-last-of-type(n)</code>	The $n^{\text{th}}$ from the last element of the parent of the specified type
<code>:only-of-type</code>	An element that has no siblings of the same type
<code>:lang(code)</code>	The element that has the specified language indicated by <code>code</code>
<code>:not(selector)</code>	An element not matching the specified <code>selector</code>

For example, the `first-of-type` pseudo-class identifies the first element of a particular type. The following selector uses this `first-of-type` pseudo-class to select the first list item found within an unordered list:

```
ul > li:first-of-type
```

This selector will not select any other list item and it will not select the first list item if it is not part of an unordered list.

Alison would like to modify the marker images used with the list of classes on the home page. Currently the `runicon.png` image file is used as the marker for all three list items. Instead, she would like to use the `runicon.png` image only for the first item, the `bikeicon.png` image as the marker for the second list item, and the `swimicon.png` as the third and last item's marker. You can use the `first-of-type`, `nth-of-type`, and `last-of-type` pseudo-classes to match the appropriate `png` file with each item.

### To apply pseudo-classes to an unordered list:

- 1. Return to the **tss\_styles.css** file in your editor.
- 2. Go to the List Styles section at the bottom of the style sheet, delete the `article#about_tss ul` style rule that sets the list style image marker and replace it with the following three style rules:

```
article#about_tss ul li:first-of-type {
 list-style-image: url(runicon.png);
}

article#about_tss ul li:nth-of-type(2) {
 list-style-image: url(bikeicon.png);
}

article#about_tss ul li:last-of-type {
 list-style-image: url(swimicon.png);
}
```

Figure 2–44 highlights the three selectors and their associated style rules using pseudo-classes with the unordered list items.

**Figure 2–44** Applying pseudo-classes to list items



- 3. Save your changes to the file and then reload the `tss_home.html` file in your browser. Figure 2–45 shows the new format of the unordered list with different image markers used with each of the list items.

Figure 2-45

## List marker images for each item

**Classes**

Winter instruction starts soon. Get a jump on your summer goals by joining us for individual or group instruction in:

- runicon.png image 
- Running:** We start with the basics to help you run faster and farther than you ever thought possible without aches and pains.
- bikeicon.png image 
- Cycling:** The indoor bike trainers at TSS include everything you need to refine your technique, stamina, and power for improved results on the road.
- swimicon.png image 
- Swimming:** The open water swim can be one of the most frightening sports to master. Our classes begin with basic techniques so that your swim can be very enjoyable, and not a chore.

© Courtesy Patrick Carey

## INSIGHT

Exploring the *nth-of-type* Pseudo-class

The *nth-of-type* pseudo-class is a powerful tool for formatting groups of elements in cyclical order. Cycles are created using the selector

`nth-of-type (an+b)`

where *a* is the length of the cycle, *b* is an offset from the start of the cycle, and *n* is a counter, which starts at 0 and increases by 1 through each iteration of the cycle. For example, the following style rules create a cycle of length 3 with the first list item displayed in red, the second displayed in blue, and the third displayed in green, after which the cycle repeats red-blue-green until the last item is reached:

```
li:nth-of-type(3n+1) {color: red;}
li:nth-of-type(3n+2) {color: blue;}
li:nth-of-type(3n+3) {color: green;}
```

When the cycle length is 1, the *nth-of-type* selector selects elements after the specified offset has passed. The following style rule sets the text color to blue for all list items starting from the 5<sup>th</sup> item

```
li:nth-of-type(n+5) {color: blue;}
```

CSS also supports the keywords *even* and *odd* so that two-length cycles can be more compactly entered as

```
li:nth-of-type(even) {color: red;}
li:nth-of-type(odd) {color: blue;}
```

with a red font applied to the even-numbered list items and a blue font applied to the odd-numbered items.

The same cyclical methods described above can be applied to the *nth-child* selector with the important difference that the *nth-child* selector selects any child element of the parent while the *nth-of-type* selector only selects elements of a specified type.

## Pseudo-classes for Hypertext

Another type of pseudo-class is a **dynamic pseudo-class** in which the class can change state based on the actions of the user. Dynamic pseudo-classes are used with hypertext links such as the `visited` class, which indicates whether the target of the link has already been visited by the user. Figure 2–46 describes the dynamic pseudo-classes.

Figure 2–46

### Dynamic pseudo-classes

Pseudo-Class	Description
<code>:link</code>	The link has not yet been visited by the user.
<code>:visited</code>	The link has been visited by the user.
<code>:active</code>	The element is in the process of being activated or clicked by the user.
<code>:hover</code>	The mouse pointer is hovering over the element.
<code>:focus</code>	The element is receiving the focus of the keyboard or mouse pointer.

For example, to display all previously visited links in a red font, you could apply the following style rule to the `a` element:

```
a:visited {color: red;}
```

To change the text color to blue when the mouse pointer is hovered over the link, apply the following rule:

```
a:hover {color: blue;}
```

#### TIP

The `hover`, `active`, and `focus` pseudo-classes also can be applied to non-hypertext elements to create dynamic page elements that change their appearance in response to user actions.

In some cases, two or more pseudo-classes can apply to the same element. For example, a hypertext link can be both visited previously and hovered over. In such situations, the standard cascading rules apply with the pseudo-class listed last applied to the element. As a result, you should enter the hypertext pseudo-classes in the following order—`link`, `visited`, `hover`, and `active`. The `link` pseudo-class comes first because it represents a hypertext link that has not been visited yet. The `visited` pseudo-class comes next, for links that have been previously visited. The `hover` pseudo-class follows, for the situation in which a user has moved the mouse pointer over a hypertext link prior to clicking the link. The `active` pseudo-class is last, representing the exact instant in which a link is activated.

Users with disabilities might interact with hypertext links through their keyboard rather than through a mouse pointer. Most browsers allow users to press the Tab key to navigate through the list of hypertext links on the page and to activate those links by pressing the Enter key. A link reached through the keyboard has the focus of the page and most browsers will indicate this focus by displaying an outline around the linked text. You can substitute your own style by using the `focus` pseudo-class in the same way that you used the `hover` pseudo-class.

#### REFERENCE

### Using Dynamic Pseudo-Class to Create Hypertext

- To create a rollover for a hypertext link, use the pseudo-classes

```
a:link
a:visited
a:hover
a:active
```

where the `link` pseudo-element matches unvisited link, `visited` matches previously visited links, `hover` matches links that have the mouse pointer hovering over them, and `active` matches links that are in the action of being clicked.

The default browser style is to underline all hypertext links; displaying the links in a blue font with previously visited links in purple. Alison wants the links in the navigation list to appear in a medium gray font with no distinction between unvisited and previously visited links. She does not want the hypertext underlined in the navigation list except when the link is hovered over or active. She also wants hovered or active links to appear in purple. Add these style rules to the style sheet now.

### To apply pseudo-classes to a hypertext links:

- 1. Return to the **tss\_styles.css** file in your editor.
- 2. Go to the Navigation Styles section and add the following style rules for hypertext links that have been visited or not visited.

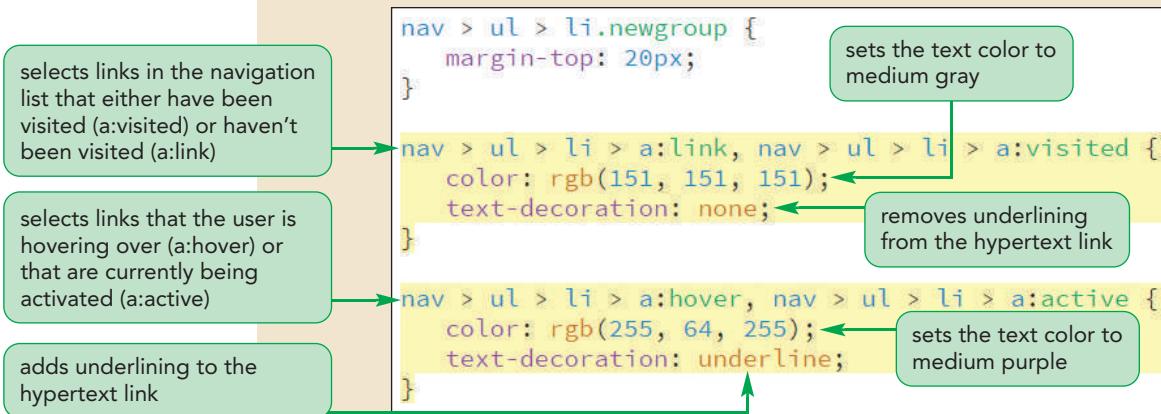
```
nav > ul > li > a:link, nav > ul > li > a:visited {
 color: rgb(151, 151, 151);
 text-decoration: none;
}
```

- 3. Add the following new style rules for links that are being hovered over or are active:

```
nav > ul > li > a:hover, nav > ul > li > a:active {
 color: rgb(255, 64, 255);
 text-decoration: underline;
}
```

Figure 2–47 highlights the style rules for hypertext links in the navigation list.

**Figure 2–47** Using pseudo-classes with hypertext links



- 4. Save your changes to the file and then reload the **tss\_home.html** file in your browser and hover your mouse pointer over the links in the navigation list. Figure 2–48 shows the hover effect applied to the link to the TSS swimming class.

Figure 2-48

Style applied to a hovered link



## PROSKILLS

### Problem Solving: Hover with Touch Devices

The hover pseudo-class was written to apply only to user interfaces that support mice or similar pointing devices. Technically, there is no hover event with touch devices, such as mobile phones and tablets. However, most mobile devices will still respond to a hover style by briefly applying the style when the user initially touches a hypertext link.

Many mobile devices also apply a “double tap” response so that initially touching a page element invokes the hover style and then immediately tapping the page element a second time invokes the click event. This technique is most often used for web pages that use the hover event to reveal hidden menus and page objects.

With the increasing importance of touch devices, a good guiding principle is that you should avoid making support for the hover style necessary for the end-user. Hover effects should be limited to enhancing the user experience but they should not be a critical component of that experience.

## Pseudo-Elements

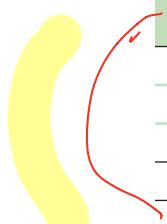
Another type of pseudo selector is a **pseudo-element**, which is an object that exists only in the rendered page. For example, a paragraph is an element that is marked in the HTML file, but the first line of that paragraph is not. Similarly, the first letter of that paragraph is also not a document element, but it certainly can be identified as an object in the web page. Pseudo-elements can be selected using the following CSS selector

`element::pseudo-element`

where *element* is an element from the HTML file and *pseudo-element* is the name of a CSS pseudo-element. Figure 2-49 describes the pseudo-elements supported in CSS.

Figure 2-49

Pseudo-elements



Pseudo-Element	Description
<code>::first-letter</code>	The first letter of the element text
<code>::first-line</code>	The first line of the element text
<code>::before</code>	Content inserted directly before the element
<code>::after</code>	Content inserted directly after the element

For example, the following style rule matches the first displayed line of every paragraph in the rendered web page and transforms the text of that line to uppercase letters:

```
p::first-line {text-transform: uppercase;}
```

The following style rule matches the first letter of every paragraph within a block quote and displays the character in a Times New Roman font that is 250% larger than the surrounding text:

```
blockquote p::first-letter {
 font-family: 'Times New Roman', Times, serif;
 font-size: 250%;
}
```

The double colon separator “::” was introduced in CSS to differentiate pseudo-elements from pseudo-classes. Older browsers use the single colon “:” for both pseudo-elements and pseudo-classes.

## Generating Content with CSS

Another type of pseudo-element is used to generate content for the web page. New content can be added either before or after an element using the following before and after pseudo-elements

```
element::before {content: text;}
element::after {content: text;}
```

where *text* is the content to be inserted into the rendered web page. The *content* property supports several types of text content as described in Figure 2–50.

Figure 2–50

Values of the content property

Value	Description
none	Sets the content to an empty text string
counter	Displays a counter value
attr( <i>attribute</i> )	Displays the value of the selector's <i>attribute</i>
text	Displays the specified <i>text</i>
open-quote	Displays an opening quotation mark
close-quote	Displays a closing quotation mark
no-open-quote	Removes an opening quotation mark, if previously specified
no-close-quote	Removes a closing quotation mark, if previously specified
url( <i>url</i> )	Displays the content of the media (image, video, etc.) from the file located at <i>url</i>

For example, the following style rules combine the *before* and *after* pseudo-elements with the *hover* pseudo-class to insert the “<” and “>” characters around every hypertext link in a navigation list:

### TIP

You cannot use CSS to insert HTML markup tags, character references, or entity references. Those can only be done within the HTML file.

```
nav a:hover::before {content: "<";}
nav a:hover::after {content: ">";}
```

Note that these style rules use both the *hover* pseudo-class and the *before/after* pseudo-elements so that the content is only inserted in response to the *hover* event.

If you want to insert a special symbol, you have to insert the code number for that symbol using text string “\code” where *code* is the code number. For example, if instead of single angled brackets as indicated above, you wanted to show double

angled brackets, « and », you would need to use the Unicode character code for these characters, 00ab and 00bb respectively. To insert these characters before and after a navigation list hypertext link, you would apply the following style rules:

```
nav a:hover::before {content: "\00ab";}
nav a:hover::after {content: "\00bb";}
```

In addition to adding content to an element as just discussed, you can also insert content that is a media file, such as an image or video clip, by using the following content property

```
content: url(url);
```

where *url* is the location of the media file. For example, the following style rule appends the image file uparrow.png to any hypertext link in the document when it is hovered over:

```
a:hover::after {content: url(uparrow.png);}
```

An image file or any content generated by the style sheet should not consist of material that is crucial to understanding your page. Instead, generated content should only consist of material that supplements the page for artistic or design-related reasons. If the generated content is crucial to understanding the page, it should be placed in the HTML file in the first place.

## Displaying Attribute Values

The content property can also be used to insert an attribute value into the rendered web page through the use of the following `attr()` function

```
content: attr(attribute);
```

where *attribute* is an attribute of the selected element. One application of the `attr()` function is to add the URL of any hypertext link to the link text. In the following code, the value of the `href` attribute is appended to every occurrence of text marked with the `a` element:

```
a::after {
 content: " (" attr(href) ")";
}
```

Notice that URL is enclosed within opening and closing parentheses. Thus, a hypertext link in an HTML document, such as

```
Triathlons
```

will be displayed in the rendered web page as:

Triathlons (http://www.triathlon.org)

This technique is particularly useful for printed output in which the author wants to have the URLs of all links displayed on the printed page for users to read and have as references.

### Inserting Content using CSS

- To insert content directly before a page element, use the style rule

```
element::before {content: text;}
```

where *element* is the page element and *text* is the content to be inserted before the element.

- To insert content directly after a page element, use the style rule

```
element::after {content: text;}
```

## Inserting Quotation Marks

The `blockquote` and `q` elements are used for quoted material. The content of these elements is usually placed in quotation marks and, while you can insert these quotation marks within the HTML file, you can also insert decorative opening and closing quotation marks using the `content` property with the following values:

```
content: open-quote;
content: close-quote;
```

The actual characters used for the open and closing quotation marks are defined for the selector with the following `quotes` property

```
quotes: "open1" "close1" "open2" "close2" ...;
```

where `open1` is the character used for the opening quotation mark and `close1` is character used for the closing quotation mark. The text strings `open2`, `close2`, and so on are used for nested quotation marks. In the example that follows, character codes are used to define the curly quotes for opening and closing quotation marks

```
quotes: "\201C" "\201D" "\2018" "\2019";
```

### TIP

Quotation marks generated by CSS are often used with international pages in which different languages require different quotation mark symbols.

where the character code 201C returns the opening curly double quote “, the code 201D returns the closing curly double quote ”, the code 2018 returns the nested opening single quote ‘, and 2019 provides the closing single quote ’.

Alison suggests that you use decorative quotes for the customer comments on the Tri and Succeed Sports home page. You display curly quotes in a bold Times New Roman font with a font size of 1.6em (which is slightly bigger than the font size of the block quote text.)

### To insert quotes into block quotes:

- 1. Return to the `tss_styles.css` file in your editor.
- 2. Go to the Aside and Blockquote Styles section and, within the style rule for the `aside blockquote` selector, add the following `quotes` property to use curly quotes for the quotation marks:

```
quotes: "\201C" "\201D";
```

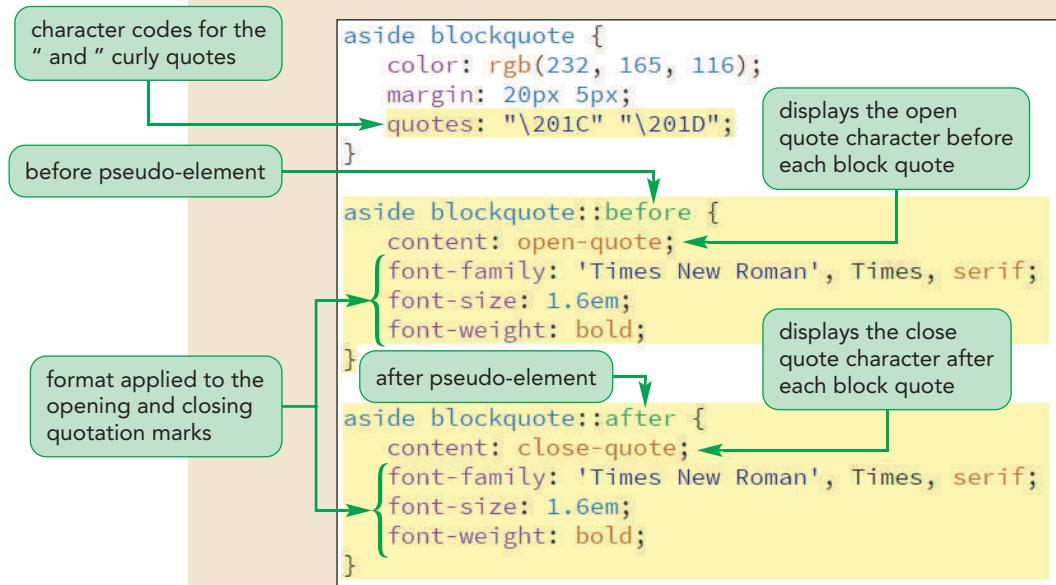
- 3. Add the following style rules to insert quotation marks before and after each block quote in the `aside` element:

```
aside blockquote::before {
 content: open-quote;
 font-family: 'Times New Roman', Times, serif;
 font-size: 1.6em;
 font-weight: bold;
}

aside blockquote::after {
 content: close-quote;
 font-family: 'Times New Roman', Times, serif;
 font-size: 1.6em;
 font-weight: bold;
}
```

Figure 2–51 highlights the styles to add curly quotes before and after each block quote.

Figure 2–51 Adding quotation marks to block quotes



- 4. Save your changes to the file and then reload the `tss_home.html` file in your browser. As shown in Figure 2–52, bold quotation marks have been added before and after each customer comment.

Figure 2–52

Quotation marks added to reviewer comments



## Validating Your Style Sheet

As with your HTML code, you can submit your CSS style sheet for validation, either to a web content management system or to the validation site at the W3C. You decide to test your CSS code for possible errors by using a validator. First add a deliberate error to the code so you can see how the CSS validator works.

### To introduce an error to the `tss_styles.css` file:

1. Scroll up and change the background color value for the `html` element from `hsl(27, 72%, 72%)` to `hsl(27, 72, 72)` (deleting the % symbols from the saturation and lightness values).
2. Save your changes to the file.

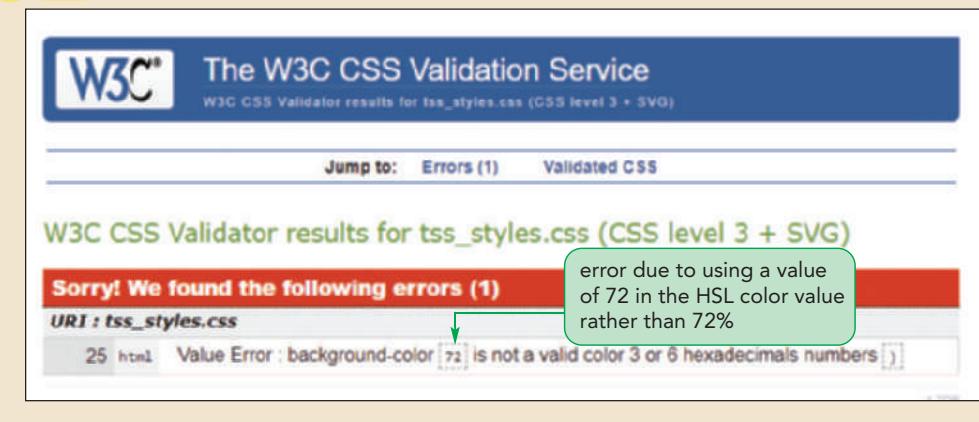
Using the validator at the W3C website or one built into your editor, validate the `tss_styles.css` file to see the impact of this change.

### To validate the code in the `tss_styles.css` file:

1. Open your browser to the W3C validator at <https://jigsaw.w3.org/css-validator/>.
2. Click the **Validate by File Upload** or **By file upload** tab within the web page.
3. Click the **Choose File** or **Browse** button in the web form and locate the `tss_styles.css` file from the `html02 ▶ tutorial` folder. Select the file and click **Open** to load the file into the validator.
4. Click the **Check** button to validate the file. See Figure 2–53.

Figure 2–53

### CSS validation report



Saturation and lightness in the HSL color value must be entered as percentages, such as 72%, rather than numbers, such as 72. The validator caught this mistake and reported it as an error. Note that unlike HTML errors, browsers are very unforgiving about CSS errors. If you make a mistake in your CSS files, the browser will ignore the style rule with the syntax error and resort to its own browser style sheet if necessary to render the page object.

### To fix the error in the `tss_styles.css` file:

1. Return to the `tss_styles.css` file in your editor.
2. Change the background color value for the `html` element back to `hsl(27, 72%, 72%)`.
3. Save your changes to the file.
4. Return in your browser to the W3C validator at <https://jigsaw.w3.org/css-validator/> or run the validator with your HTML editor.
5. Retest the `tss_styles.css` file, verifying that no errors or warnings are reported.



### Teamwork: Managing a Style Sheet

Your style sheets often will be as long and as complex as your website content. As the size of a style sheet increases, you might find yourself overwhelmed by multiple style rules and definitions. This can be an especially critical problem in a workplace where several people need to interpret and sometimes edit the same style sheet. Good management skills are as crucial to good design as a well-chosen color or typeface are. As you create your own style sheets, here are some techniques to help you manage your creations:

- Use style comments throughout, especially at the top of the file. Clearly describe the purpose of the style sheet, where it's used, who created it, and when it was created.
- Because color values are not always immediately obvious, include comments that describe your colors. For example, annotate a color value with a comment such as "body text is tan".
- Divide your style sheet into sections, with comments marking the section headings.
- Choose an organizing scheme and stick with it. You may want to organize style rules by the order in which they appear in your documents, or you may want to insert them alphabetically. Whichever you choose, be consistent and document the organizing scheme in your style comments.
- Keep your style sheets as small as possible, and break them into separate files if necessary. Use one style sheet for layout, another for text design, and perhaps another for color and graphics. Combine the style sheets using the `@import` rule, or combine them using the `link` element within each page. Also, consider creating one style sheet for basic pages on your website, and another for pages that deal with special content. For example, an online store could use one style sheet (or set of sheets) for product information and another for customer information.

By following some of these basic techniques, you'll find your style sheets easier to manage and develop, and it will be easier for your colleagues to collaborate with you to create an eye-catching website.

Alison is pleased with the work you've done on the typography and design of the Tri and Succeed Sports website. Alison will continue to develop the new version of the website and will get back to you with future changes and design ideas.

### Session 2.3 Quick Check

1. Which style will display a list with lowercase letters as the list marker?
  - a. `list-style-type: lower-letter;`
  - b. `list-style-case: lower;`
  - c. `list-style: lower-case;`
  - d. `list-style-type: lower-alpha;`
2. Which style rule will display all unordered lists using the star.png image file placed inside the virtual box?
  - a. `ul { list-style-image: star.png; list-position: inside; }`
  - b. `ul { list-style-image: url(star.png); list-style-position: inside; }`

- c. `ul { list-image: url(star.png); list-position: inside; }`
- d. `ul { list-style-listImage: star.png; inside: true; }`
3. Which style displays all previously visited hypertext links in gray?  
 a. `a:visited { color: gray; }`  
 b. `a:previousVisit { color: gray; }`  
 c. `a:link { color: gray; }`  
 d. `a#visited { color: gray; }`
4. What is the size of the left padding space in the style property padding-spacing: 10px 5px 8px; ?  
 a. 10px **top right bottom left**  
 b. 5px  
 c. 8px  
 d. The size is not specified
5. Which of the following styles sets the size of the left margin to 30 pixels?  
 a. `margin: 30px;`  
 b. `margin: 10px 30px;`  
 c. `margin: 10px 30px 5px;`  
 d. All of the above
6. The first line of a paragraph is referenced using which selector?  
 a. `p#first-line;`  
 b. `p.first-line;`  
 c. `p > first-line;`  
 d. `p::first-line;`
7. The space at the outer extent of an element is found within the element's  
 a. content  
 b. padding  
 c. border  
 d. margin
8. Which style rule will insert the text string \*\*\*\* before every paragraph belonging to the Reviews class?  
 a. `p#Reviews {content: ****;}`  
 b. `p#Reviews {before: ****;}`  
 c. `p.Reviews.before {content: ****;}`  
 d. `p.Reviews::before {content: ****;}`
9. Which style rule will underline the last item in an ordered list?  
 a. `ol li:last {text-decoration: underline;}`  
 b. `ol li:last-of-type {text-decoration: underline;}`  
 c. `ol li#last {text-decoration: underline;}`  
 d. `ol li.last-of-type {text-decoration: underline;}`

ol lis : last-of-type

**OBJECTIVES****Session 3.1**

- Create a reset style sheet
- Explore page layout designs
- Center a block element
- Create a floating element
- Clear a floating layout
- Prevent container collapse

**Session 3.2**

- Use CSS grid styles
- Define a grid layout
- Place items within a grid
- Work with grid areas

**Session 3.3**

- Explore positioning styles
- Work with relative positioning
- Work with absolute positioning
- Work with overflow content

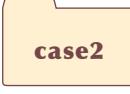
# Designing a Page Layout

## *Creating a Website for a Chocolatier*

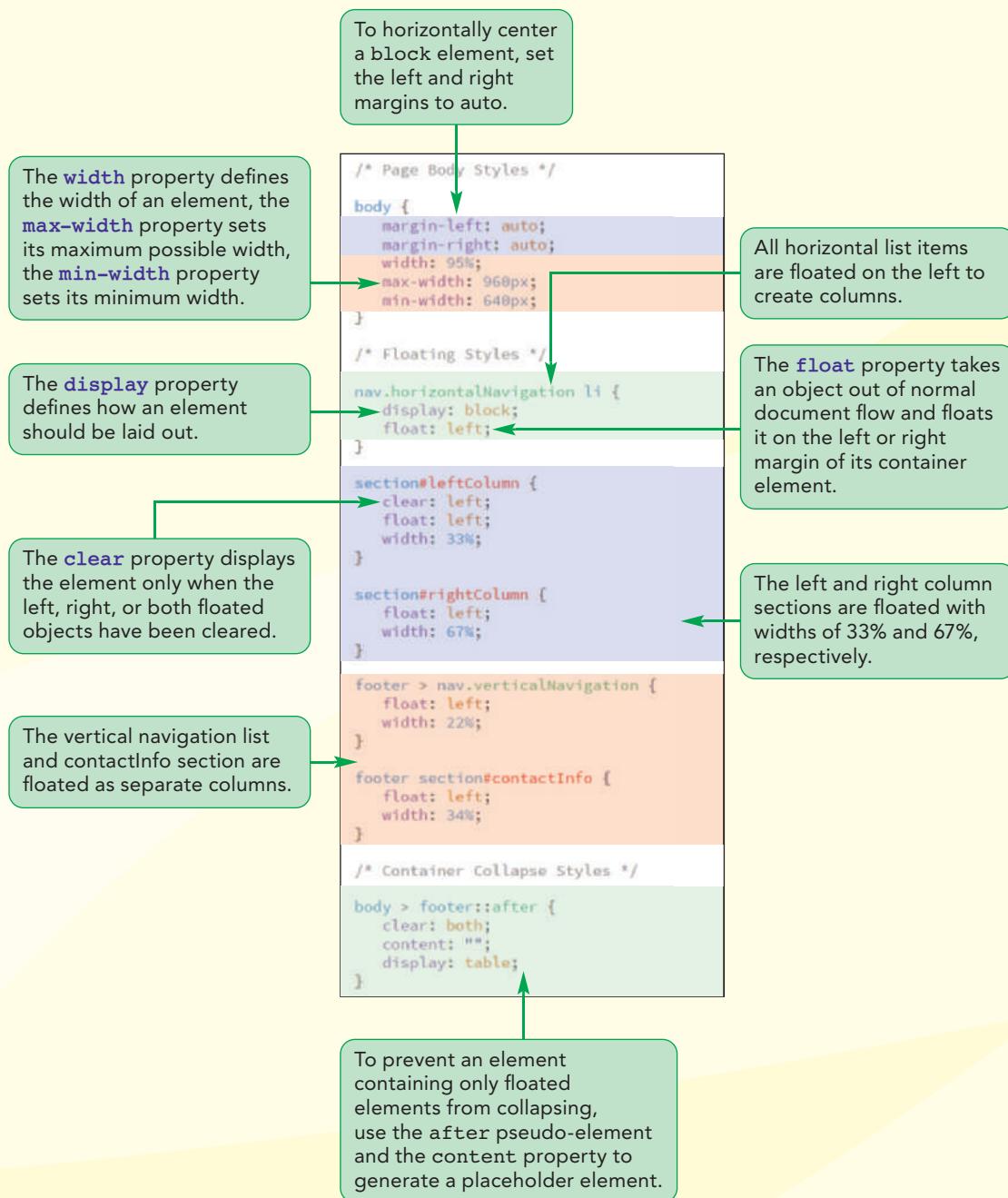
### **Case | Pandaisia Chocolates**

Anne Ambrose is the owner and head chocolatier of *Pandaisia Chocolates*, a chocolate shop located in Essex, Vermont. You have been asked to assist on the redesign of the company's website. Anne has provided you with three pages from the website to start your work. She has written all of the content, compiled the necessary images and graphics, and written some of the text and color styles. She needs you to complete the project by designing the page layout using the CSS layout properties.

### STARTING DATA FILES

 <b>html03</b>	 <b>tutorial</b>	 <b>review</b>	 <b>code1</b>
	pc_about_txt.html pc_home_txt.html pc_info_txt.html pc_grids_txt.css pc_home_txt.css pc_reset_txt.css + 22 files	pc_specials_txt.html pc_specials_txt.css + 12 files	code3-1_txt.html code3-1_float_txt.css + 11 files
 <b>code2</b>	 <b>code3</b>	 <b>code4</b>	
	code3-2_txt.html code3-2_layout_txt.css + 1 file	code3-3_txt.html code3-3_scroll_txt.css + 10 files	code3-4_txt.html debug3-4_txt.css + 2 files
 <b>case1</b>	 <b>case2</b>	 <b>demo</b>	
	sp_home_txt.html sp_layout_txt.css + 13 files	ss_dday_txt.html ss_layout_txt.css + 4 files	demo_grid1.html demo_grid2.html demo_positioning.html + 6 files

# Session 3.1 Visual Overview:



# Page Layout with Floating Elements

Page body is horizontally centered within the browser window.

414 Tree Lane • Essex, VT 05452

[Home](#) [Online Store](#) [My Account](#) [Specials](#) [Contact Us](#)

Pandaisia Chocolates has been creating gourmet chocolates and sweets for happy customers since 1993. Our hand-dipped truffles, savory chocolates, and mouth-watering toffees are made from the finest organic cacao. We use only natural ingredients with wildflower honey replacing corn syrup and cream that comes fresh from local dairy farms. Chocolate is an art: come tour our gallery.

[Chocolate](#) [Fudge](#) [Toffee](#) [Truffles](#)

**The Store**  
[About Us](#)  
[Facebook](#)  
[Twitter](#)  
[Reviews](#)  
[Info-graphic](#)

**Products**  
[Online Store](#)  
[Gift Boxes](#)  
[Collections](#)  
[Weddings](#)  
[Specials](#)

**Services**  
[My Account](#)  
[Order History](#)  
[Tracking](#)  
[Privacy Policy](#)  
[Contact Us](#)

**Location & Hours**  
 414 Tree Lane  
 Essex, Vermont 05452  
 (802) 555-0414  
 Mon - Thu: 9 a.m. - 8 p.m.  
 Sat - Sun: 9 a.m. - 5 p.m.  
 Toll Free: 1-800-555-0414

Horizontal list items are floated into separate columns.

Left and right sections are floated into separate columns.

The contents of the page footer are floated into separate columns.

© Brenda Carson/Shutterstock.com;  
 © Brent Hofacker/Shutterstock.com;  
 © Jim Bowie/Shutterstock.com;  
 © wacomka/Shutterstock.com;  
 © Shebeko/Shutterstock.com;  
 Source: Facebook;  
 Source: Twitter, Inc.

## Introducing the display Style

The study of page layout starts with defining how an individual element is presented on the page. In the first tutorial, you learned that HTML elements are classified into block elements, such as paragraphs or headings, or into inline elements, such as emphasized text or inline images. However, whether an element is displayed as a block or as inline depends on the style sheet. You can define the display style for any page element with the following `display` property

```
display: type;
```

where `type` defines the display type. A few of the many `type` values are shown in Figure 3–1.

Figure 3–1

Some values of the `display` property

Display Value	Appearance
<code>block</code>	Displayed as a block
<code>table</code>	Displayed as a web table
<code>inline</code>	Displayed inline within a block
<code>inline-block</code>	Treated as a block placed inline within another block
<code>run-in</code>	Displayed as a block unless its next sibling is also a block, in which case, it is displayed inline, essentially combining the two blocks into one
<code>inherit</code>	Inherits the <code>display</code> property of the parent element
<code>list-item</code>	Displayed as a list item along with a bullet marker
<code>none</code>	Prevented from displaying, removing it from the rendered page

For example, to supersede the usual browser style that displays images inline, you can apply the following style rule to display all of your images as blocks:

```
img {display: block;}
```

If you want to display all block quotes as list items, complete with list markers, you can add the following style rule to your style sheet:

```
blockquote {display: list-item;}
```

### TIP

You also can hide elements by applying the style `visibility: hidden;`, which hides the element content but leaves the element still occupying the same space in the page.

You can even prevent browsers from displaying an element by setting its `display` property to `none`. In that case, the element is still part of the document structure but it is not shown to users and does not occupy space in the displayed page. This is useful for elements that include content that users shouldn't see or have no need to see.

You'll use the `display` property in creating a reset style sheet.

## Creating a Reset Style Sheet

You learned in the last tutorial that your browser applies its own styles to your page elements unless those styles are superseded by your own style sheet. Many designers prefer to work with a “clean slate” and not have any browser style rules creep into the final design of their website. This can be accomplished with a `reset style sheet` that supersedes the browser's default styles and provides a consistent starting point for page design.

You'll create a reset style sheet for the Pandaisia Chocolates website. The first style rule in your sheet will use the `display` property to display all of the HTML 5 structural elements in your web page as blocks. While current browsers already do this, there are some older browsers that do not recognize or have predefined display styles for elements as such `header`, `article`, or `footer`. By including the `display` property in a reset style sheet, you add a little insurance that these structural elements will be rendered correctly.

### To create a reset style sheet:

- 1. Use the text editor or HTML editor of your choice to open the `pc_reset_txt.css` file from the `html03 ▶ tutorial` folder. Enter **your name** and **the date** in the comment section of the file and save the document as `pc_reset.css`.
- 2. Within the Structural Styles section, insert the following style rule to define the display properties of several HTML 5 structural elements:

```
article, aside, figcaption, figure,
footer, header, main, nav, section {
 display: block;
}
```

Figure 3–2 highlights the new style rule in the document.

Figure 3–2

### Displaying structural elements as blocks

```
/* Structural Styles */

article, aside, figcaption, figure,
footer, header, main, nav, section {
 display: block;
}
```

You will complete the reset style sheet by adding other style rules that set default padding and margins around commonly used page elements, define some basic typographic properties, and remove underlining from hypertext links found within navigation lists.

### To complete the reset style sheet:

- 1. Within the Typographic Styles section, insert the following style rule to define the typographic styles for several page elements:

```
address, article, aside, blockquote, body, cite,
div, dl, dt, dd, em, figcaption, figure, footer,
h1, h2, h3, h4, h5, h6, header, html, img,
li, main, nav, ol, p, section, span, ul {

 background: transparent;
 font-size: 100%;
 margin: 0;
 padding: 0;
 vertical-align: baseline;
}
```

- 2. Add the following style rules to remove list markers from list items found within navigation lists:
 

```
nav ul {
 list-style: none;
 list-style-image: none;
}

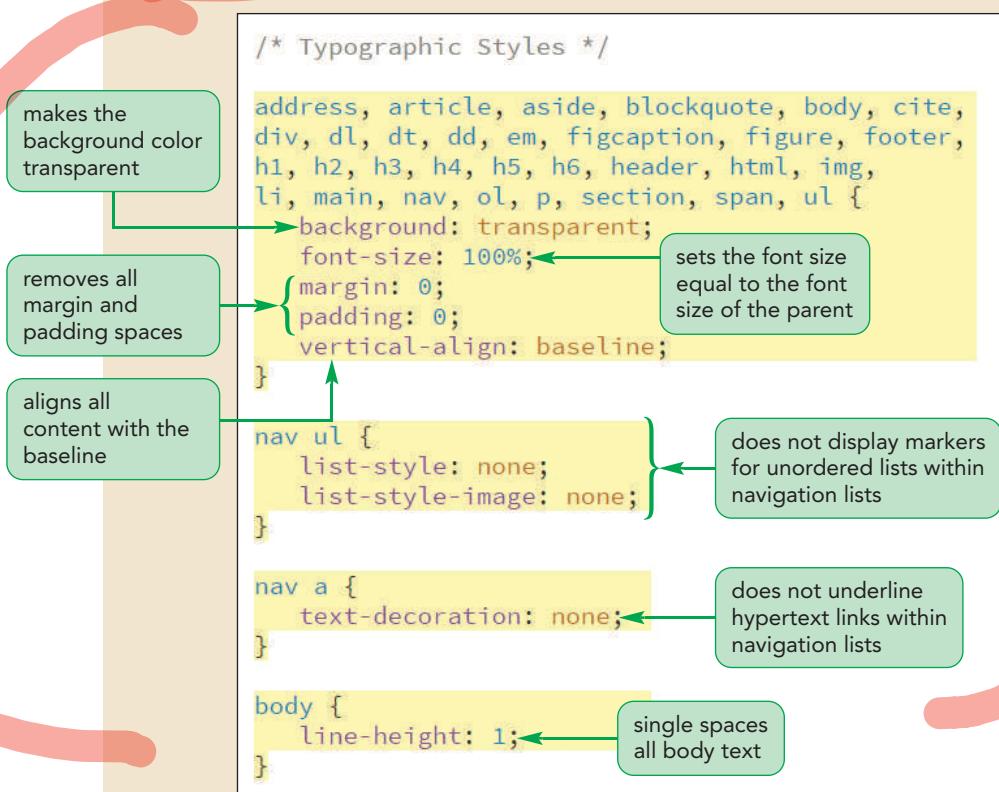
nav a {
 text-decoration: none;
}
```
- 3. Set the default line height to 1 (single-spaced) by applying the following style rule to the page body:
 

```
body {
 line-height: 1;
}
```

Figure 3–3 describes the new style rules in the document.

Figure 3–3

### Completing the reset style sheet



- 4. Save your changes to the file.

This is a very basic reset style sheet. There are premade reset style sheets freely available on the web that contain more style rules used to reconcile the various differences between browsers and devices. Before using any of these reset style sheets, you should study the CSS code and make sure that it meets the needs of your website. Be aware that some reset style sheets may contain more style rules than you actually need and you can speed up your website by paring down the reset sheet to use only the elements you need for your website.

The first page you will work on for Pandaisia Chocolates is the site's home page. Anne has already created a typographical style sheet in the pc\_styles1.css file. Link to the style sheet file now as well as the pc\_reset.css style sheet you just created and the pc\_home.css style sheet that you will work on for the remainder of this session to design the page layout.

### To get started on the Pandaisia Chocolates home page:

- 1. Use your editor to open the **pc\_home\_txt.css** file from the html03 ▶ tutorial folder. Enter **your name** and **the date** in the comment section of the file and save the document as **pc\_home.css**.
- 2. Use your editor to open the **pc\_home\_txt.html** file from the same folder. Enter **your name** and **the date** in the comment section and save the file as **pc\_home.html**.
- 3. Within the document head, directly after the `title` element, insert the following `link` elements to link the home page to the **pc\_reset.css**, **pc\_styles1.css** and **pc\_home.css** style sheets:

```
<link href="pc_reset.css" rel="stylesheet" />
<link href="pc_styles1.css" rel="stylesheet" />
<link href="pc_home.css" rel="stylesheet" />
```
- 4. Take some time to study the content and structure of the **pc\_home.html** document. Pay particular attention to the use of ID and class names throughout the document.
- 5. Save your changes to the file. You might want to keep this file open as you work with the **pc\_home.css** style sheet so that you can refer to its content and structure.

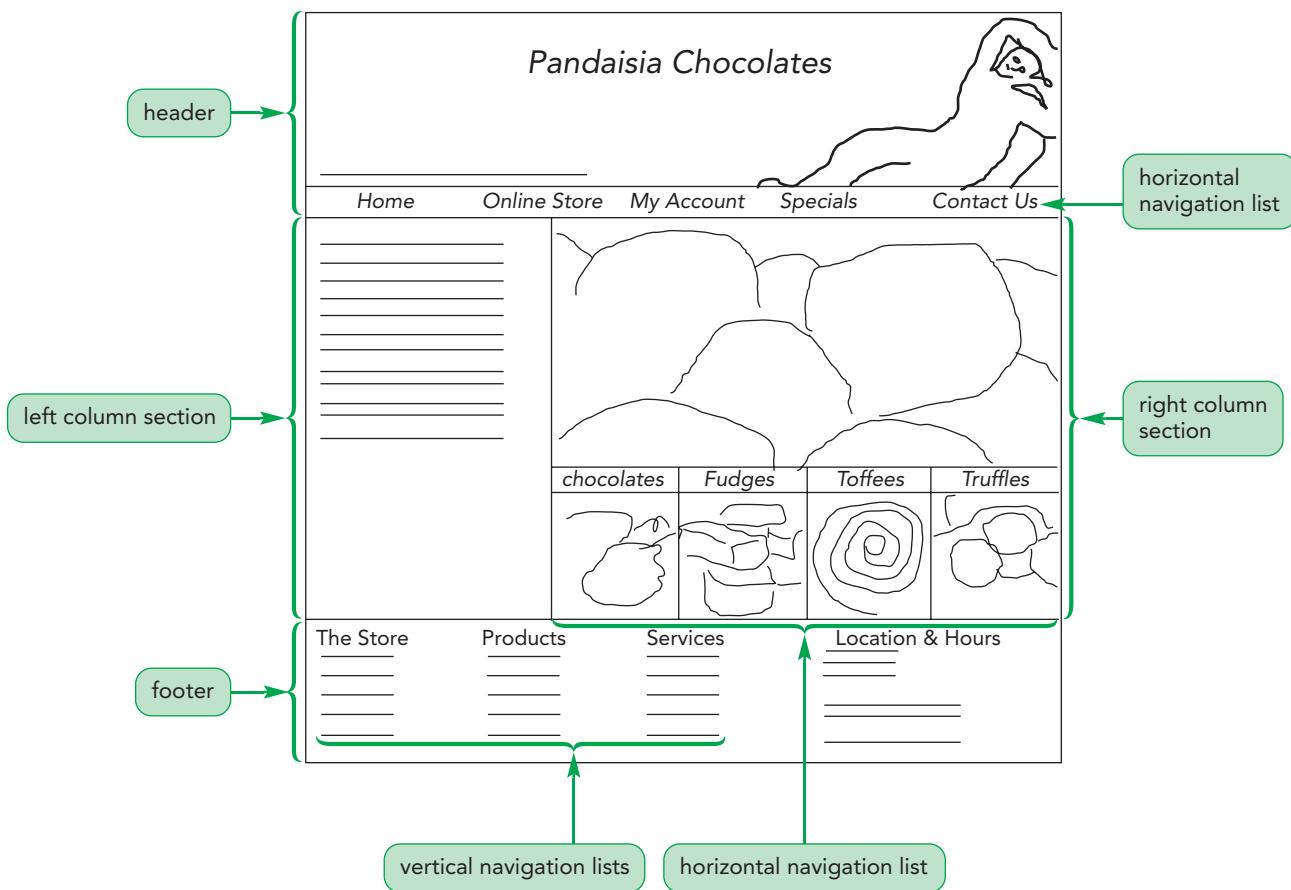
#### TIP

The reset style sheet should always be the first style sheet listed before any other style sheets to ensure that your default styles are applied first.

Anne has sketched the general layout she wants for the home page, shown in Figure 3–4. Compare the **pc\_home.html** file content to the sketch shown in Figure 3–4 to get a better understanding of how the page content relates to Anne's proposed layout.

Figure 3-4

Proposed home page layout



Before creating the page layout that Anne has sketched out for you, you'll examine different types of layout designs.

## Exploring Page Layout Designs

One challenge of layout is that your document will be viewed on many different devices with different screen resolutions. When designing for the web, you're usually more concerned about the available screen width than screen height because users can scroll vertically down the length of the page, but it is considered bad design to make them scroll horizontally.

A page designer needs to cope with a wide range of possible screen widths ranging from wide screen monitors with widths of 1680 pixels or more, down to mobile devices with screen widths of 320 pixels and even less. Complicating matters even more is that a screen width represents the maximum space available to the user, but some space is always taken up by toolbars, sidebar panes, and other browser features. In addition, the user might not even have the browser window maximized to fill the entire screen. Thus, you need a layout plan that will accommodate a myriad of screen resolutions and browser configurations.

### Fixed, Fluid, and Elastic Layouts

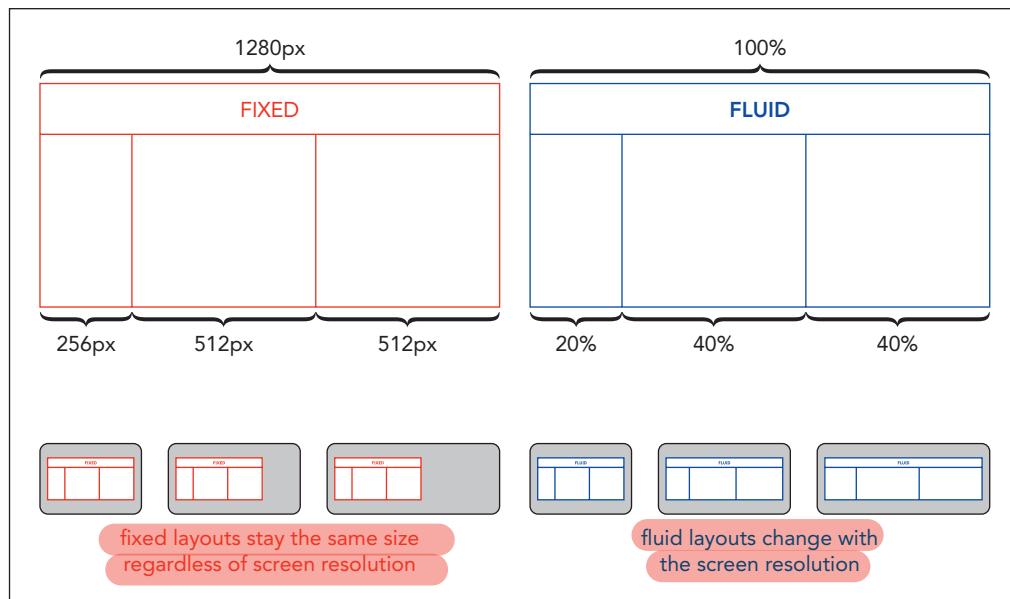
Web page layouts fall into three general categories: fixed, fluid, and elastic. A **fixed layout** is one in which the size of the page and the size of the page elements are fixed, usually using pixels as the unit of measure. The page width might be set at 960 pixels and the

width of the company logo set to 780 pixels. These widths are set regardless of the screen resolution of the user's device and this can result in the page not fitting into the browser window if the device's screen is not wide enough.

By contrast, a **fluid layout** sets the width of page elements as a percent of the available screen width. For example, the width of the page body might be set to fill 90% of the screen and the width of the company logo might be set to fill 80% of that page body. Under a fluid layout, the page resizes automatically to match the screen resolution of the user's device. Figure 3–5 shows how a three-column layout might appear in both a fixed and a fluid design.

Figure 3–5

### Fixed layouts vs. fluid layouts



With different devices accessing your website, it's usually best to work with a fluid layout that is more adaptable to a range of screen resolutions. Fixed layouts should only be used when you have more control over the devices that will display your page, such as a web page created specifically for a digital kiosk at a conference.

Another layout design is an **elastic layout** in which all measurements are expressed in em units and based on the default font size used in the page. If a user or the designer increases the font size, then the width, height, and location of all of the other page elements, including images, change to match. Thus, images and text are always sized in proportion to each other and the layout never changes with different font sizes. The disadvantage to this approach is that, because sizing is based on the font size and not on the screen resolution, there is a danger that if a user sets the default font size large enough, the page will extend beyond the boundaries of the browser window.

Finally, the web is moving quickly toward the principles of **responsive design** in which the layout and design of the page change in response to the device that is rendering it. The page will have one set of styles for mobile devices, another for tablets, and yet another for laptops or desktop computers. You'll explore how to implement responsive design in Tutorial 5.

Because width is such an integral part of layout, you will start designing the Pandasia Chocolates home page by defining the width of the page body and elements within the page.

## Working with Width and Height

The width and height of an element are set using the following `width` and `height` properties

 `width: value;` `height: value;` *→ using percentage*

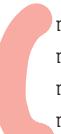
where `value` is the width or height using one of the CSS units of measurement or as a percentage of the width or height of the parent element. For example, the following style rule sets the width of the page body to 95% of the width of its parent element (the browser window):

```
body {width: 95%;}
```

Usually, you do not set the `height` value because browsers automatically increase the height of an element to match its content. Note that all block elements, like the `body` element, have a default width of 100%. Thus, this style rule makes the `body` element width slightly smaller than it would be by default.

## Setting Maximum and Minimum Dimensions

You can set limits on the width or height of a block element by applying the following properties

 `min-width: value;`  
`min-height: value;`  
`max-width: value;`  
`max-height: value;` *→ using pixels*

where `value` is once again a length expressed in one of the CSS units of measure (usually pixels to match the measurement unit of the display device). For example, the following style rule sets the width of the page body to 95% of the browser window width but confined within a range of 640 to 1680 pixels:

 `body {`  
    `width: 95%;`  
    `min-width: 640px;`  
    `max-width: 1680px;`  
`}`

Maximum and minimum widths are often used to make page text easier to read. Studies have shown that lines of text that are too wide are difficult to read because the eye has to scan across a long section of content and that lines of text that are too narrow with too many line returns break the flow of the material.

### Setting Widths and Heights

- To set the width and height of an element, use the styles

```
width: value;
height: value;
```

where *value* is the width or height in one of the CSS units of measurement or a percentage of the width or height of the parent element.

- To set the minimum possible width or height, use the styles

```
min-width: value;
min-height: value;
```

- To set the maximum possible width or height, use the styles

```
max-width: value;
max-height: value;
```

Set the width of the page body for the Pandaisia Chocolates home page to 95% of the browser window ranging from 640 pixels to 960 pixels. Also display the company logo image as a block with its width set to 100% so that it extends across the page body. You do not have to set the height of the logo because the browser will automatically scale the height to keep the original proportions of the image.

#### To set the initial dimensions of the page:

- 1. Return to the **pc\_home.css** file in your editor and add the following style rule to the Body Styles section:

```
body {
 max-width: 960px;
 min-width: 640px;
 width: 95%;
}
```

- 2. Within the Body Header Styles section, insert the following style rule to set the display type and width of the logo image:

```
body > header > img {
 display: block;
 width: 100%;
}
```

Figure 3–6 highlights the newly added style rules in the style sheet.

Figure 3–6

## Setting the width of the page body and logo

```

/* Body Styles */

body {
 max-width: 960px;
 min-width: 640px;
 width: 95%;
}

/* Body Header Styles */

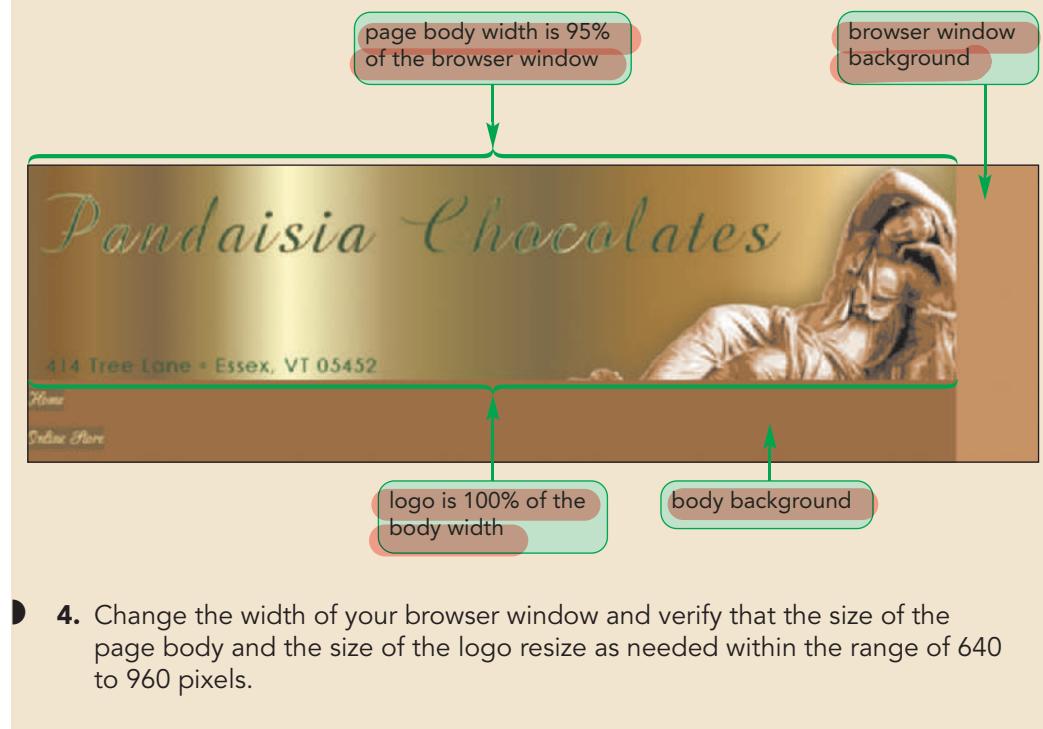
body > header > img {
 display: block;
 width: 100%;
}

```

- 3. Save your changes to the file and then open the **pc\_home.html** file in your browser. Figure 3–7 shows the current layout of the page body and logo.

Figure 3–7

## Initial view of the body header



- 4. Change the width of your browser window and verify that the size of the page body and the size of the logo resize as needed within the range of 640 to 960 pixels.

The page body is currently placed on the left margin of the browser window. Anne would like it centered horizontally within the browser window.

## Centering a Block Element

Block elements can be centered horizontally within their parent element by setting both the left and right margins to `auto`. Thus, you can center the page body within the browser window using the style rule:

*margin: value1 value2;  
top/bottom left/right*

`body {`

`margin-left: auto;  
margin-right: auto;`

*) → centered horizontally*

Modify the style rule for the page body to center the Pandaisia Chocolates home page horizontally by setting the left and right margins to `auto`.

### To center the page body horizontally:

- 1. Return to the `pc_home.css` file in your editor and, within the style rule for the `body` selector, insert the properties:

```
margin-left: auto;
margin-right: auto;
```

Figure 3–8 highlights the newly added styles.

Figure 3–8

Centering the page body

```
body {
 margin-left: auto;
 margin-right: auto;
 max-width: 960px;
 min-width: 640px;
 width: 95%;
}
```

setting the left and right margins to `auto` forces block elements to be horizontally centered within their parent

- 2. Save your changes to the file and then reload the `pc_home.html` file in your browser. Verify that the page body is now centered within the browser window.

## INSIGHT

### Working with Element Heights

The fact that an element's height is based on its content can cause some confusion. For example, the following style rule appears to set the height of the header to 50% of the height of the page body:

```
body > header {height: 50%;}
```

However, because the total height of the page body depends on the height of its individual elements, including the body header, there is circular reasoning in this style rule. You can't set the page body height without knowing the height of the body header and you can't set the body header height unless you know the height of the page body. Most browsers deal with this circularity by leaving the body header height undefined, resulting in no change in the layout.

Heights need to be based on known values, as in the following style rules where the body height is set to 1200 pixels and thus the body header is set to half of that or 600 pixels.

```
body {height: 1200px;}
body > header {height: 50%;}
```

It is common in page layout design to extend the page body to the height of the browser window. To accomplish this, you set the height of the `html` element to 100% so that it matches the browser window height (a known value defined by the physical properties of the screen) and then you set the minimum height of the page body to 100% as in the following style rules:

```
html {height: 100%;}
body {min-height: 100%;}
```

The result is that the height of the page body will always be at least equal to the height of the browser window, but it will extend beyond that if necessary to accommodate extra page content.

## Vertical Centering

Centering an element vertically within its parent element is not easily accomplished because the height of the parent element is usually determined by its content, which might not be a defined value. One solution is to display the parent element as a table cell with a defined height and then set the `vertical-align` property set to `middle`. For example, to vertically center the following `h1` heading within the `div` element

```
<div>
 <h1>Pandaisia Chocolates</h1>
</div>
```

you would apply the style rule:

```
div {
 height: 40px;
 display: table-cell;
 vertical-align: middle;
}
```

Using this style rule, the `h1` heading will be vertically centered.

To vertically center a single line of text within its parent element, set the line height of the text larger than the text's font size. The following style rule will result in an h1 heading with vertically centered heading text.

```
h1 {
 font-size: 1.4em;
 line-height: 2em;
}
```

Note that this approach will only work for a single line of text. If the text wraps to a second line, it will no longer be vertically centered. Vertical centering is a common design challenge and there are several other workarounds that have been devised over the years. The simplest approach is to use CSS grid styles, a topic that we'll discuss in the next session.

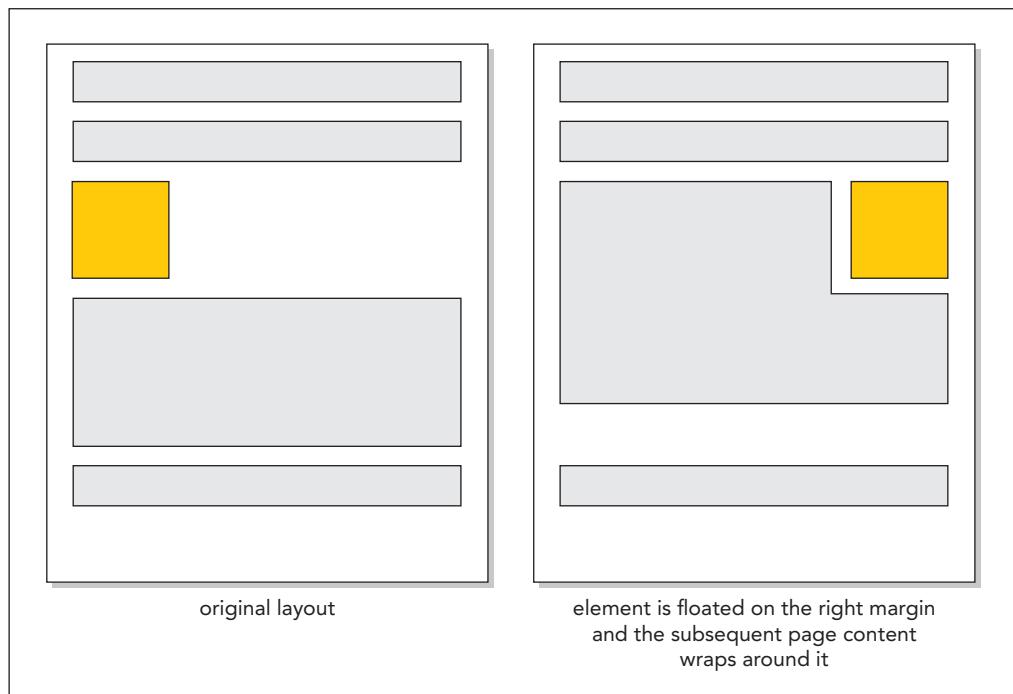
Next, you will lay out the links in the navigation list. Anne wants the links displayed horizontally rather than vertically. You can accomplish this using CSS floats.

## Floating Page Content

By default, content is displayed in the page in the order it appears within the HTML file as part of the normal document flow. **Floating** an element takes it out of position and places it along the left or right edge of its parent element. Subsequent content that is not floated occupies the space previously taken up by the floated element. Figure 3–9 shows a diagram of an element that is floated along the right margin of its container and its effect on the placement of subsequent content.

Figure 3–9

Floating an element



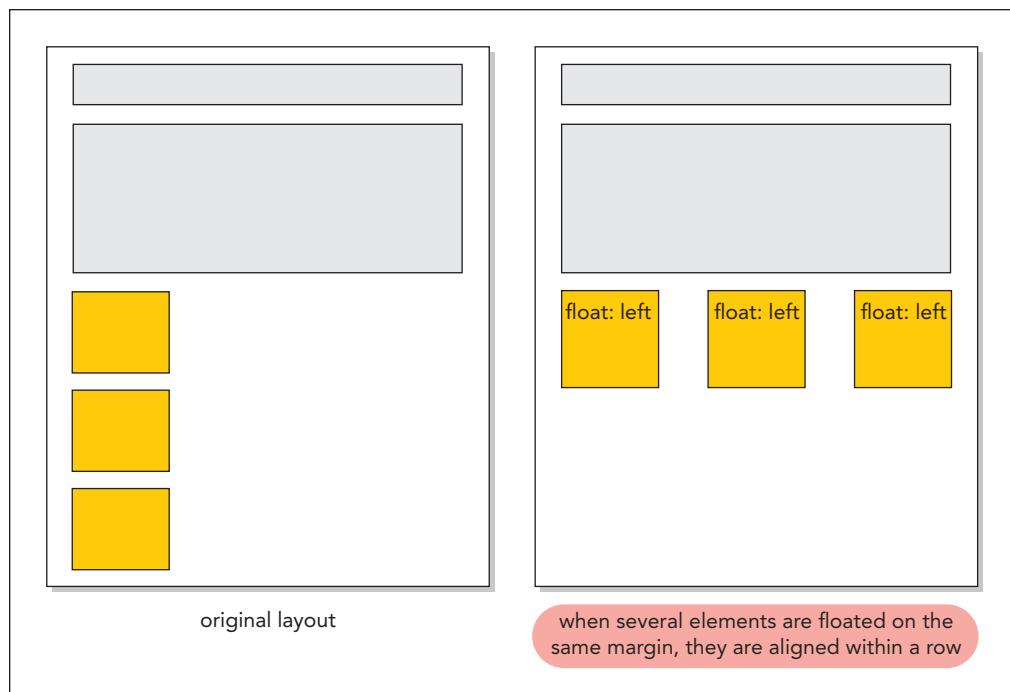
To float an element, apply the following `float: position;`

```
float: position;
```

where `position` is `none` (the default), `left` to float the object on the left margin, or `right` to float the object on the right margin. If sibling elements are floated along the same margin, they are placed alongside each other within a row as shown in Figure 3–10.

Figure 3–10

Floating multiple elements in a row



Note that for the elements to be placed within a single row, the combined width of the elements cannot exceed the total width of their parent element, otherwise any excess content will automatically wrap to a new row.

## REFERENCE

### Floating an Element

- To float an element within its container, apply the style `float: position;` where `position` is `none` (the default), `left`, or `right`.

Anne wants you display the content of navigation lists belonging to the `horizontalNavigation` class within a single row. You will accomplish this by floating each item in those navigation lists on the left margin using the `float` property. Create this style rule now.

### To lay out horizontal navigation list items:

- 1. Return to the `pc_home.css` file in your editor and go to the Body Header Styles section.
- 2. Because there are five links in the navigation list, you'll make each list item 20% of the width of the navigation list by adding the following style rule:

```
body > header > nav.horizontalNavigation li {
 width: 20%;
}
```

To be confined to a single row, the total width of floated elements cannot exceed the width of the container.

3. Insert the following style rule within the Horizontal Navigation Styles section to display every list item within a horizontal navigation list as a block floated on the left.

```
nav.horizontalNavigation li {
 display: block;
 float: left;
}
```

Figure 3-11 highlights the styles used with list items.

Figure 3-11

### Floating items in the navigation list

```
/* Body Header Styles */

body > header > img {
 display: block;
 width: 100%;
}

body > header > nav.horizontalNavigation li {
 width: 20%; }

/* Horizontal Navigation Styles */

nav.horizontalNavigation li {
 display: block;
 float: left; }
```

sets the width of the list item to 20% of the width of the navigation list

floats the list item within every horizontal navigation list as a block on the left

4. Save your changes to the file and then reload the pc\_home.html file in your browser. Figure 3-12 shows the revised layout of the navigation list in the page header.

Figure 3-12

### Floating items in a horizontal navigation list



Anne doesn't like the appearance of the hypertext links in the navigation list. Because the links are inline elements, the background color extends only as far as the link text. She suggests you change the links to block elements and center the link text within each block.

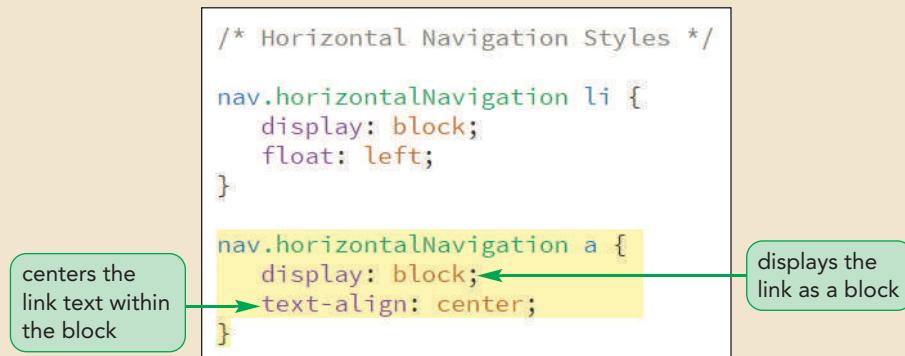
### To change the display of the hypertext links:

- 1. Return to the `pc_home.css` file in your editor.
- 2. Within the Horizontal Navigation Styles section, insert the following style rule to format the appearance of the hypertext links within the horizontal navigation lists:

```
nav.horizontalNavigation a {
 display: block;
 text-align: center;
}
```

Figure 3-13 highlights the style rule for the hypertext links.

**Figure 3-13** Formatting hyperlinks in horizontal navigation lists



- 3. Save your changes to the file and then reload the `pc_home.html` file in your browser.
- 4. Hover your mouse pointer over the links in the navigation list. Note that the link text is centered within its block and the background color extends fully across the block rather than confined to the link text. See Figure 3-14.

**Figure 3-14** Links in the body header



**Trouble?** Don't worry about the jumble of elements displayed after the body header. You'll straighten out those objects next.

You have completed the design of the body header. Next, you will lay out the middle section of the home page.

### Creating Drop Caps with CSS

**INSIGHT** A popular design element is the **drop cap**, which consists of an enlarged initial letter that drops down into a body of text. To create a drop cap, you increase the font size of an element's first letter and float it on the left margin. Drop caps also generally look better if you decrease the line height of the first letter, enabling the surrounding content to better wrap around the letter. Finding the best combination of font size and line height is a matter of trial and error, and unfortunately, what looks best in one browser might not look as good in another. The following style rule works well in applying a drop cap to the first paragraph element:

```
p:first-of-type::first-letter {
 font-size: 4em;
 float: left;
 line-height: 0.8;
}
```

For additional design effects, you can change the font face of the drop cap to a cursive or decorative font.

### Clearing a Float

In some layouts, you will want an element to be displayed on a new row, clear of previously floated objects. To ensure that an element is always displayed below your floated elements, apply the following `clear` property:

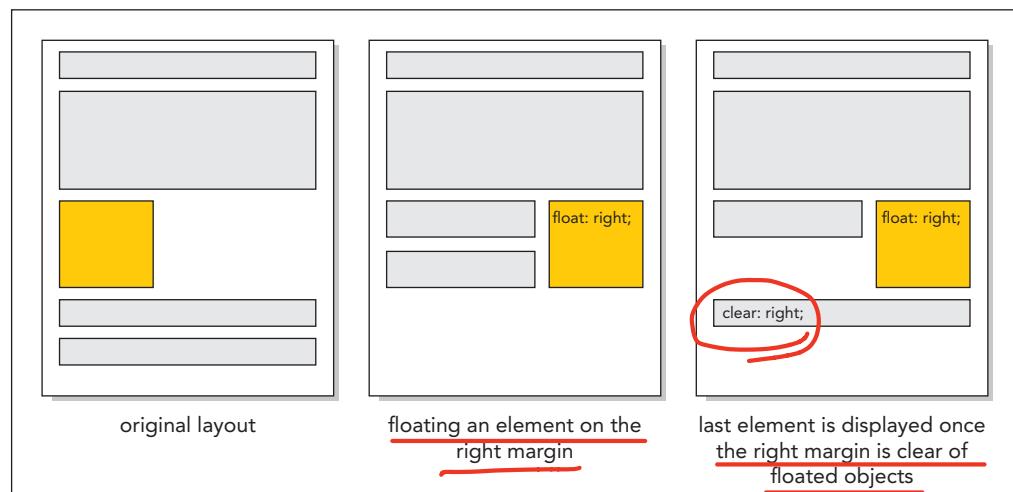
```
clear: position;
```

where `position` is `left`, `right`, `both`, or `none`. A value of `left` displays the element only when the left margin is clear of floating objects. A value of `right` displays the element only when the right margin is clear. A value of `both` displays the element only when both margins are clear of floats. The default `clear` value is `none`, which allows the element to be displayed alongside any floated objects.

Figure 3–15 shows how use of the `clear` property prevents an element from being displayed until the right margin is clear of floats. The effect on the page layout is that the element is shifted down and is free to use the entire page width since it is no longer displayed alongside a floating object.

Figure 3–15

Clearing a float



### Clearing a Float

- To display a non-floated element on a page with a floated element, use the following style so the non-floated element can clear the floated element

```
clear: position;
```

where *position* is none (the default), left, right, or both.

The next part of the Pandaisia Chocolates home page contains two `section` elements named `leftColumn` and `rightColumn`. Set the width of the left column to 33% of the body width and set the width of the right column to 67%. Float the sections side-by-side on the left margin, but only when the left margin is clear of all previously floated objects.

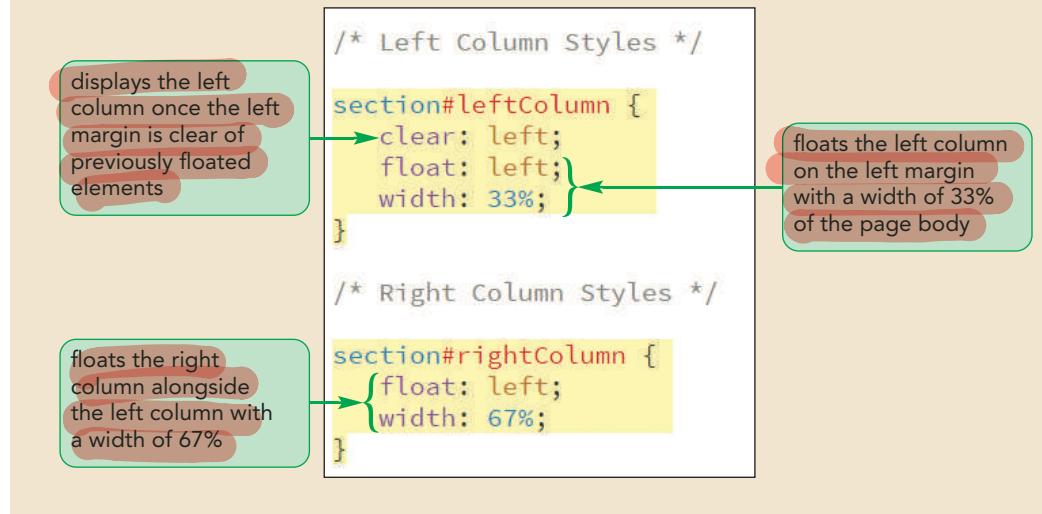
### To float the left and right column sections:

- 1. Return to the `pc_home.css` file in your editor. Go to the Left Column Styles section and insert the style rule:
- ```
section#leftColumn {
    clear: left;
    float: left;
    width: 33%;
}
```
- 2. Within the Right Column Styles section, insert:
- ```
section#rightColumn {
 float: left;
 width: 67%;
}
```

Note that you do not apply the `clear` property to the right column because you want it to be displayed in the same row alongside the left column. Figure 3–16 highlights the style rules for the left and right columns.

Figure 3–16

Float the left and right column sections



The right column contains a horizontal navigation list containing four items, each consisting of an image and a label above the image. Anne wants the four items placed side-by-side with their widths set to 25% of the width of the navigation list. Anne also wants the images in the right column displayed as blocks with their widths set to 100% of their parent element.

### To complete the right column section:

- 1. Within the Right Column Styles section, insert the following style rules to format the inline images and list items:

```
section#rightColumn img {
 display: block;
 width: 100%;
}

section#rightColumn > nav.horizontalNavigation li {
 width: 25%;
}
```

Note that you do not have to include a style rule to float the items in the horizontal navigation list because you have already created that style rule in Figure 3-11. Figure 3-17 describes the new style rules in the style sheet.

Figure 3-17 Formatting the right column section

```
/* Right Column Styles */

section#rightColumn {
 float: left;
 width: 67%;
}

section#rightColumn img {
 display: block;
 width: 100%;
}

section#rightColumn > nav.horizontalNavigation li {
 width: 25%;
}
```

sets the width of each list item to 25% of the width of the navigation list

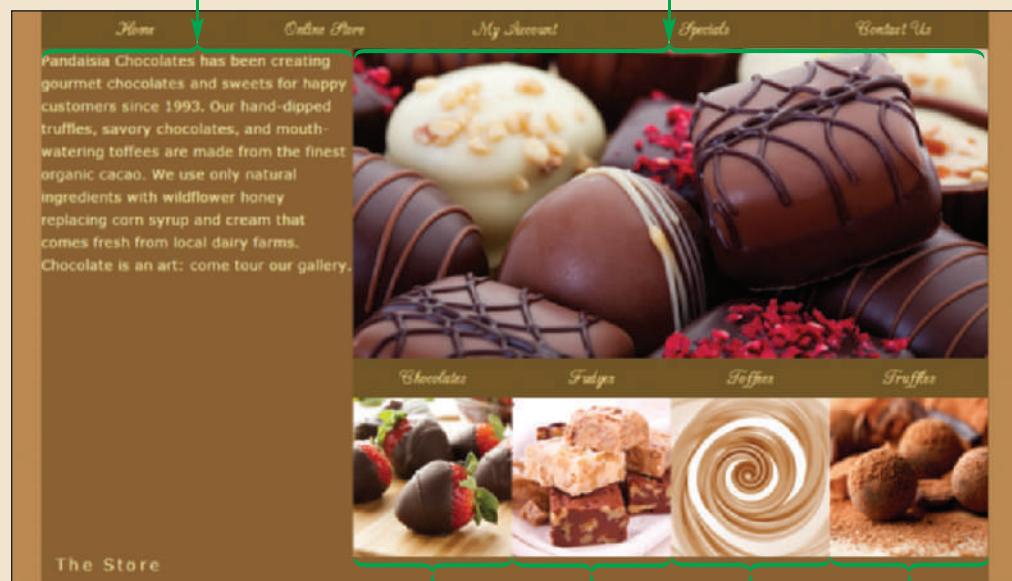
displays every image in the right column as a block with a width equal to the width of its parent element

- 2. Save your changes to the file and then reload the `pc_home.html` file in your browser. Figure 3-18 shows the layout of the left and right column sections.

Figure 3-18

Layout of the left and right columns

horizontal navigation list with each image and label set to 25% of the list width



© Brenda Carson/Shutterstock.com; © Brent Hofacker/Shutterstock.com; © Jim Bowie/Shutterstock.com; © wacomka/Shutterstock.com; © Shebeko/Shutterstock.com

Anne doesn't like that the text in the left column crowds the right column and page boundary. She suggests that you provide more interior space by increasing the padding in the left column.

### To increase the left column padding:

- 1. Return to the **pc\_home.css** file in your editor and go to the Left Column Styles section.
- 2. Insert the property **padding: 1.5em;** into the `section#leftColumn` style rule as shown in Figure 3-19.

Figure 3-19

Increasing the padding of the left column

```
/* Left Column Styles */

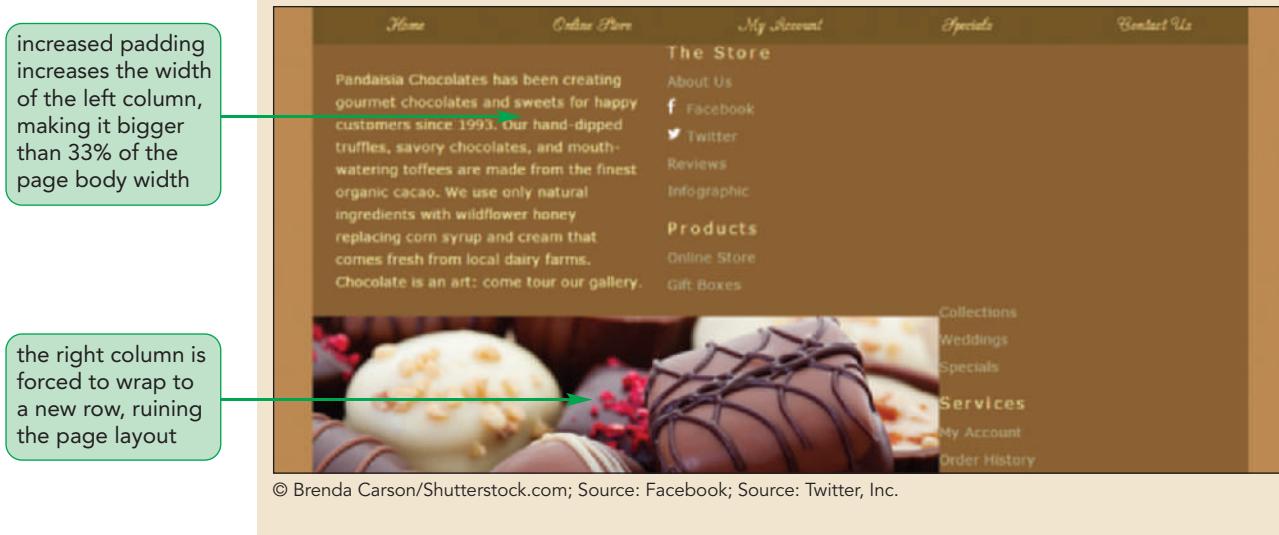
section#leftColumn {
 clear: left;
 float: left;
 padding: 1.5em; ←
 width: 33%;
```

increases the interior padding to 1.5em

- 3. Save your changes to the style sheet and then reload the `pc_home.html` file in your browser. Figure 3–20 shows the result of your change.

Figure 3–20

### Page layout crashes with increased padding



This simple change has caused the layout to crash. What went wrong?

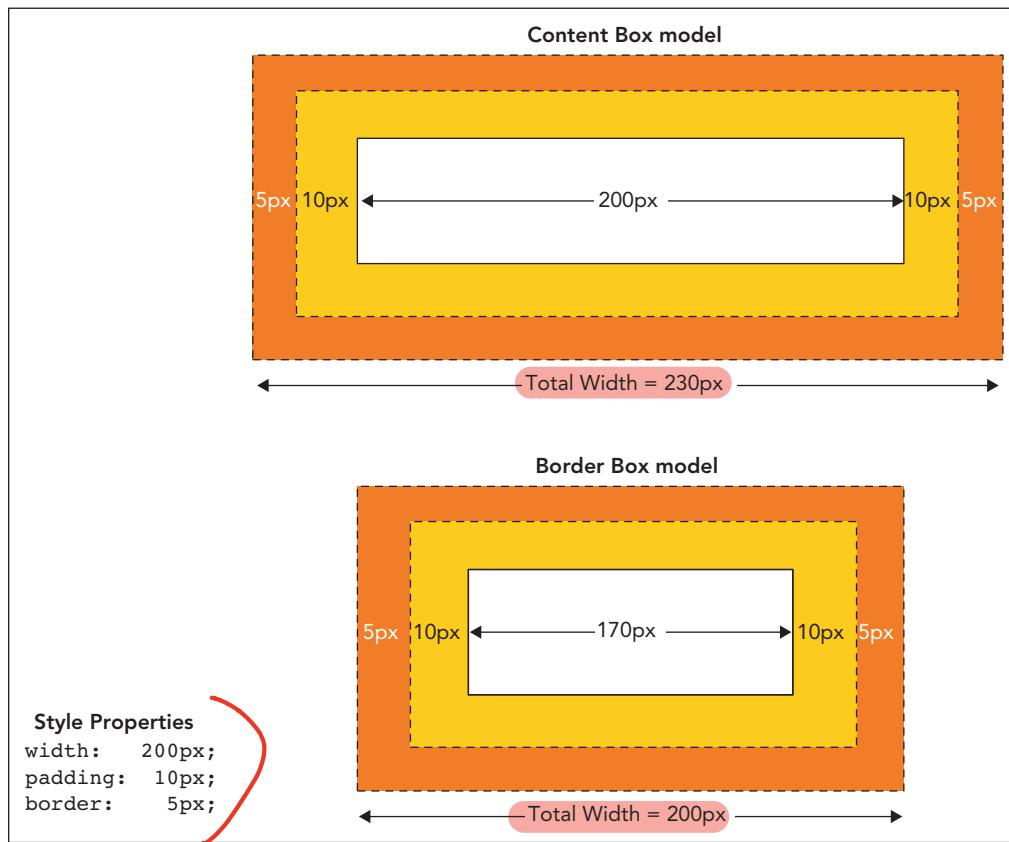
### Refining a Floated Layout

When the total width of floated objects exceeds the width of their parent, excess content is automatically wrapped to a new row. The reason the layout for the Pandaisia Chocolates home page crashed is that increasing the padding in the left column, increased the column's width beyond its set value of 33%. Even this small increase caused the total width of the two columns to exceed 100% and, as a result, the right column moved to a new row.

To keep floats within the same row, you have to understand how CSS handles widths. Recall that block elements are laid out according to the box model, as illustrated previously in Figure 2–38, in which the content is surrounded by the padding space, the border space, and finally the margin space. By default, browsers measure widths using the **content box model** in which the `width` property only refers to the width of the element content and any padding or borders constitute added space.

CSS also supports the **border box model**, in which the `width` property is based on the sum of the content, padding, and border spaces and any space taken up by the padding and border is subtracted from space given to the content. Figure 3–21 shows how the two different models interpret the same width, padding, and border values.

Figure 3–21 Comparing the content box and border box models



### TIP

Height values are similarly affected by the type of layout model used.

You can choose the layout model using the following `box-sizing` property

`box-sizing: type;`

where `type` is `content-box` (the default), `border-box`, or `inherit` (to inherit the property defined for the element's container). Many designers prefer to use the border box model in page layout so that there is no confusion about the total width of each element.

### REFERENCE

#### Defining How Widths Are Interpreted

- To define what the `width` property measures, use the style:

`box-sizing: type;`

where `type` is `content-box` (the default), `border-box`, or `inherit` (to inherit the property defined for the element's container).

Add the `box-sizing` property to the reset style sheet and apply it to all block elements.

### To set the block layout model:

- 1. Return to the **pc\_reset.css** file in your editor.
- 2. Add the following style property to the style rule for the list of block elements:

```
box-sizing: border-box;
```

Figure 3–22 highlights the revised style rule.

Figure 3–22

### Adding the border-box style to the reset style sheet

applies border-box sizing to all of the listed block elements

```
address, article, aside, blockquote, body, cite,
div, dl, dt, dd, em, figcaption, figure, footer,
h1, h2, h3, h4, h5, h6, header, html, img,
li, main, nav, ol, p, section, span, ul {
 background: transparent;
 font-size: 100%;
 margin: 0;
 padding: 0;
 vertical-align: baseline;
 box-sizing: border-box;
}
```

- 3. Save your changes to the style sheet and then reload the **pc\_home.html** file in your browser. Verify that the layout of the left and right columns has been restored and additional padding has been added within the left column.

The final part of the Pandaisia Chocolates home page is the footer, which contains three vertical navigation lists and a **section** element with contact information for the store. Once the left margin is clear of previously floated objects, float these four elements on the left margin with the widths of the three navigation lists each set to 22% of the body width and the **section** element occupying the remaining 34%.

### To lay out the page footer:

- 1. Return to the **pc\_home.css** file in your editor and scroll down to the Footer Styles section.
- 2. Insert the following style rules:

```
footer {
 clear: left;
}

footer > nav.verticalNavigation {
 float: left;
 width: 22%;
}

footer > section#contactInfo {
 float: left;
 width: 34%;
}
```

Figure 3–23 highlights the layout style rules for the page footer.

**Figure 3–23** Setting the layout of the page footer

```
/* Footer Styles */

footer {
 clear: left;
}

footer > nav.verticalNavigation {
 float: left;
 width: 22%;
}

footer > section#contactInfo {
 float: left;
 width: 34%;
```

- 3. Save your changes to the style sheet and then reload pc\_home.html in your browser. Figure 3–24 shows the new layout of the footer.

**Figure 3–24** Page footer layout



Anne asks you to change the background color of the footer to a dark brown to better show the text content.

### To set the footer background color:

- 1. Return to the `pc_home.css` file in your editor and go to the Footer Styles section.
- 2. Insert the following property for the `footer` selector:

```
background-color: rgb(71, 52, 29);
```

Figure 3–25 highlights the footer background color style.

Figure 3–25

### Setting the footer background color

footer background set to a dark brown

```
footer {
 background-color: rgb(71, 52, 29);
 clear: left;
}
```

- 3. Save your changes to the style sheet and then reload `pc_home.html` in your browser. Note that the background color is *not* changed.

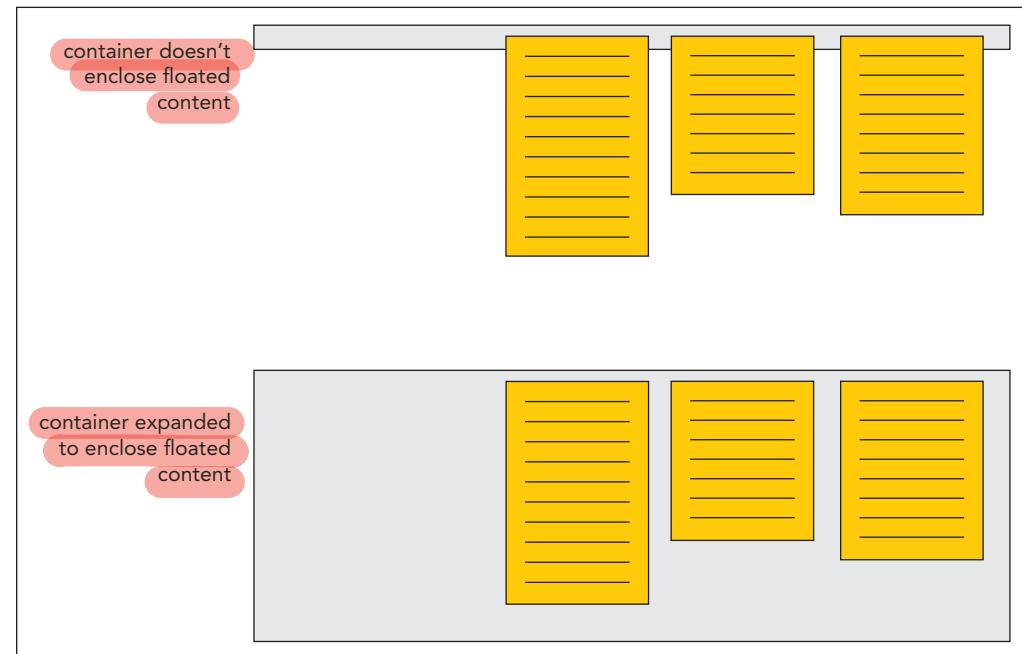
Why didn't the change to the background color take effect? To help you understand why, you'll look once again at the nature of floated elements.

## Working with Container Collapse

Recall that a floated element is taken out of the document flow so that it is no longer “part” of the element that contains it. Literally it is floating free of its container. When every element in a container is floated, there is no content left. As far as the browser is concerned, the container is empty and thus has no height and no background to color, a situation known as **container collapse**. Figure 3–26 demonstrates container collapse for a container that has three floating objects that exceed the boundaries of their container.

Figure 3–26

### Container collapse



What you usually want in your layout is to have the container expand to surround all of its floating content. One way this can occur is if the container is followed by another element that is displayed only when the margins are clear of floats. In that situation, the container's height will expand up to that trailing element and in the process surround its floating content.

The problem with the footer in the Pandaisia home page is that there is no trailing element—the footer is the last element in the page body. One way to fix that problem is to use the `after` pseudo-element to add a placeholder element after the footer. The general style rule is

```
container::after {
 clear: both;
 content: "";
 display: table;
}
```

### TIP

To find other ways to prevent container collapse, search the web using the keywords *CSS clearfix*.

where `container` is the selector for the element containing floating objects. The `clear` property keeps this placeholder element from being inserted until both margins are clear of floats. The element itself is a web table but contains only an empty text string so that no actual content is written to the web page. That's okay because the mere presence of this placeholder element is enough to keep the container from collapsing.

Add a style rule now to create a placeholder element that keeps the footer from collapsing around its floating content.

### To keep the footer from collapsing:

- 1. Return to Footer Styles section in the `pc_home.css` file and, after the style rule for the footer element, insert the following rule:

```
footer::after {
 clear: both;
 content: "";
 display: table;
}
```

Figure 3–27 highlights the new rule in the style sheet.

Figure 3–27

Preventing the footer from collapsing

```
footer {
 background-color: rgb(71, 52, 29);
 clear: left;
}

footer::after {
 clear: both;
 content: "";
 display: table;
}
```

- 2. Save your changes to the style sheet and then reload `pc_home.html` in your browser. Figure 3–28 shows the completed layout of the Pandaisia Chocolates home page.

Figure 3–28

Final layout of the Pandaisia Chocolates home page



footer has expanded to contain all floated content

© Brenda Carson/Shutterstock.com; © Brent Hofacker/Shutterstock.com; © Jim Bowie/Shutterstock.com;  
© wacomka/Shutterstock.com; © Shebeko/Shutterstock.com; Source: Facebook; Source: Twitter, Inc.

Note that the footer now has a dark brown background because it has expanded in height to contain all of its floated content.

- 3. Close any of the documents you opened for this session.

## REFERENCE

### Keeping a Container from Collapsing

- To prevent a container from collapsing around its floating content, add the following style rule to the container

```
container::after {
 clear: both;
 content: "";
 display: table;
}
```

where `container` is the selector for the element containing the floating content.



PROSKILLS

### Problem Solving: The Virtue of Being Negative

It's common to think of layout in terms of placing content, but good layout also must be concerned with placing emptiness. In art and page design, this is known as working with positive and negative space. Positive space is the part of the page occupied by text, graphics, borders, icons, and other page elements. Negative space, or white space, is the unoccupied area and provides balance and contrast to elements contained in positive space.

A page that is packed with content leaves the eye with no place to rest; which also means that the eye has no place to focus and maybe even no clear indication about where to start reading. Negative space is used to direct users to resting stops before moving on to the next piece of page content. This can be done by providing a generous margin between page elements and by increasing the padding within an element. Even increasing the spacing between letters within an article heading can alleviate eye strain and make the text easier to read.

White space also has an emotional aspect. In the early days of print advertising, white space was seen as wasted space, and thus, smaller magazines and direct mail advertisements would tend to crowd content together in order to reduce waste. By contrast, upscale magazines and papers could distinguish themselves from those publications with an excess of empty space. This difference carries over to the web, where a page with less content and more white space often feels more classy and polished, while a page crammed with a lot of content feels more commercial. Both can be effective; you should decide which approach to use based on your customer profile.

You've completed your work on the Pandaisia Chocolates home page. In the next session, you'll work on page layout using the technique of grids.

REVIEW

### Session 3.1 Quick Check

1. To display an element as a block-level use:  
  - a. `display: block-level;`
  - b. `display: block;`
  - c. `display: inline;`
  - d. `display: display-block;`
2. What are three types of layouts?  
  - a. inline, fluid, static
  - b. fixed, floating, static
  - c. fixed, fluid, elastic
  - d. inline, block, scrolling
3. Provide a style rule to set the maximum width of an element to 960 pixels.  
  - a. `maximum-width: 960px;`
  - b. `maxw: 960px;`
  - c. `width: 960px;`
  - d. `max-width: 960px;`

margin: 10px  
 5px  
 top/bottom  
 right and left

4. Provide a style rule to horizontally center a block element within its container with a top/bottom margin of 20 pixels.

- a. margin: 20px center;
- b. margin: center 20px;
- c. margin: auto 20px;
- d. margin: 20px auto;

top/bottom 20px  
 right/left auto

5. Provide a style rule to place an object on the right margin of its container.

- a. margin: right;
- b. text-align: right;
- c. float: right;
- d. padding: right;

6. Provide a style rule to display an object only when all floating elements have cleared.

- a. clear: float;
- b. clear: floats;
- c. clear: both;
- d. clear: all;

7. Your layout has four floated elements in a row but unfortunately the last element has wrapped to a new line. What is the source of the layout mistake?

- a. The widths of the floated elements exceed the available width of their container.  
 b. You cannot float more than one object within a row.  
 c. You have to clear the first three floating object to make room for the fourth.  
 d. You have to clear the fourth floating object to make room for the first three.

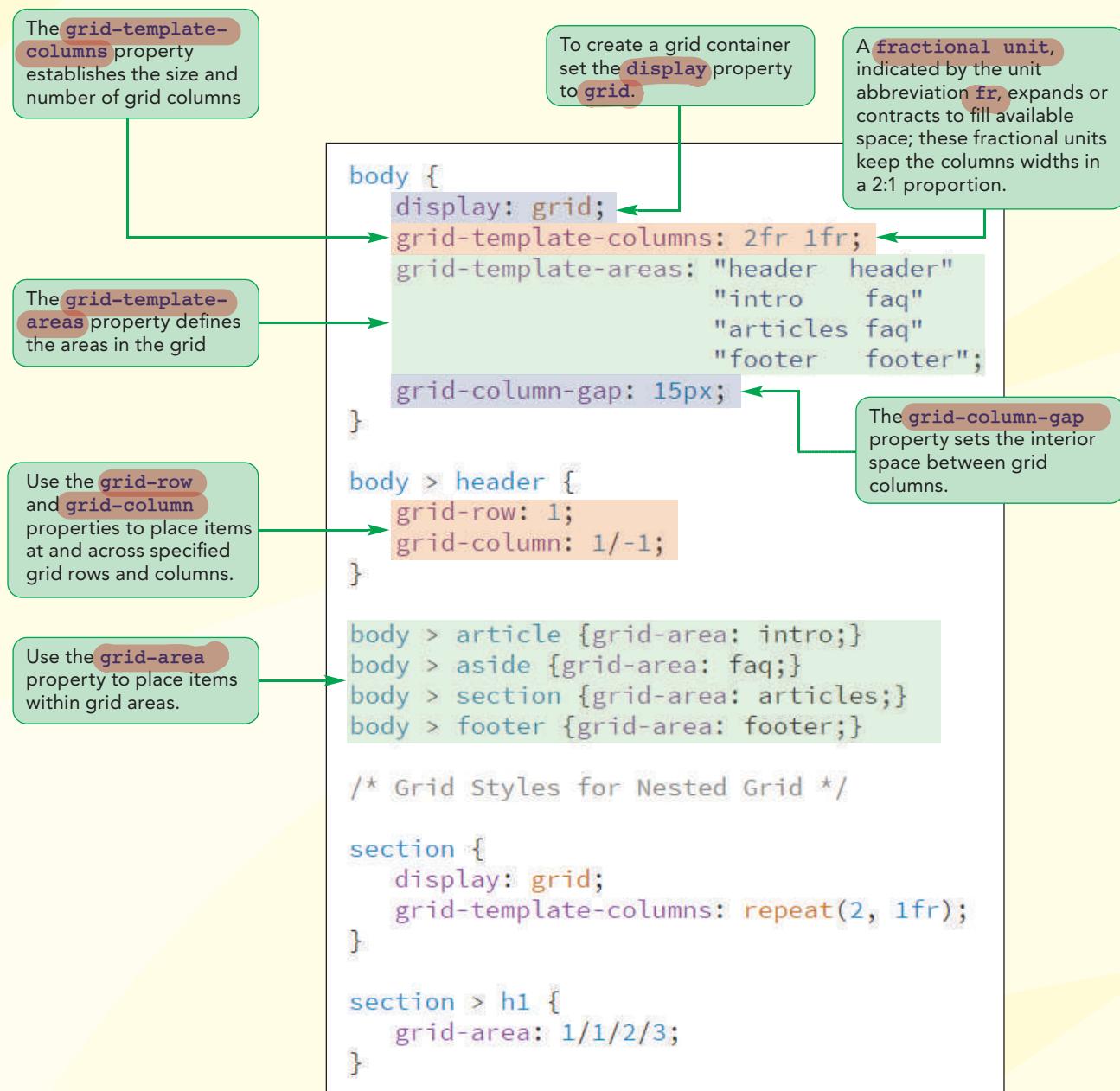
8. Provide a style rule to change the width property for the header element so that it measures the total width of the header content, padding, and border spaces.

- a. box-sizing: border-box;
- b. box-sizing: content-box;
- c. box-sizing: all;
- d. box-sizing: complete;

9. What causes container collapse?

- a. The width of the child elements exceeds the width of the container.
- b. The width of the container element is fixed at 0 pixels.
- c. The height of the container element is fixed at 0 pixels.
- d. All child elements are floating so that they are free of the container, leaving the container with no content.

## Session 3.2 Visual Overview:



# CSS Grid Layouts

Page header covers the grid from column gridline 1 to -1 (the last gridline)

Columns laid out in a proportion of 2:1

aside element placed in the FAQ grid area

footer element placed in the footer grid area

© Twin Design/Shutterstock.com

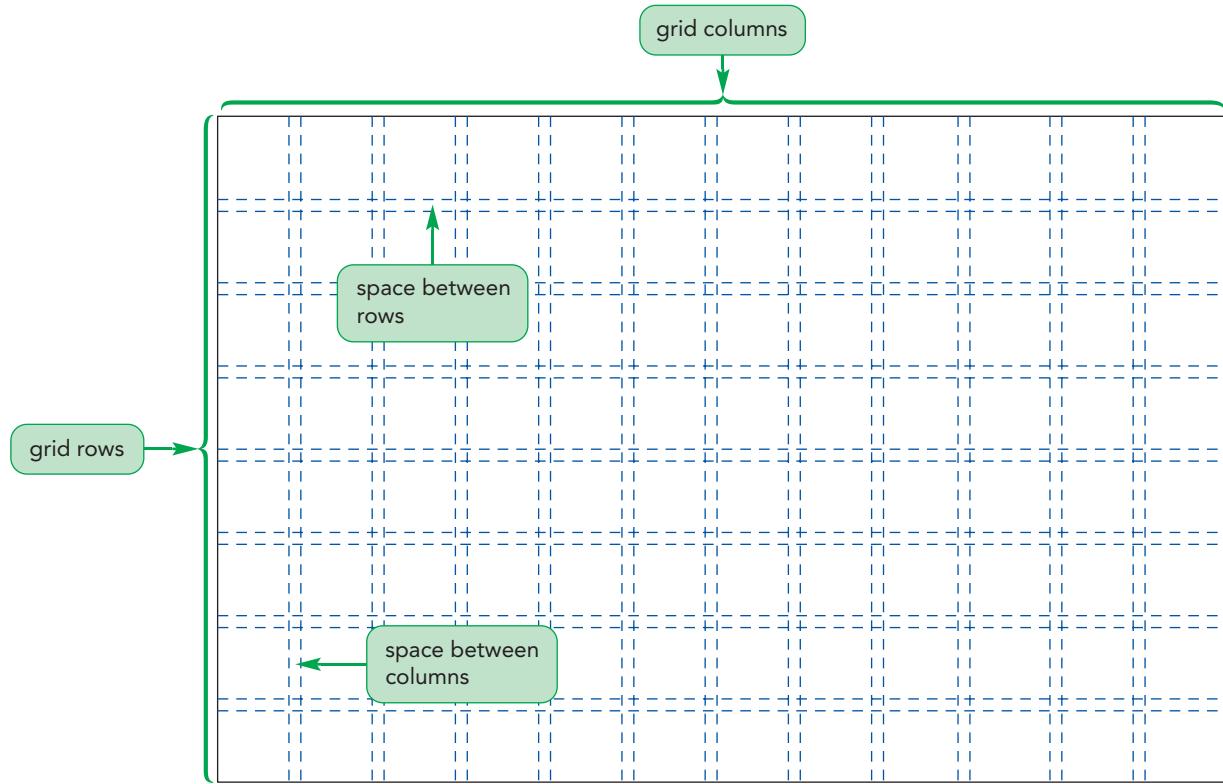
## Introducing Grid Layouts

In the previous session, you used the `float` property to lay out a page in sections that floated alongside each other like columns. In this session, you'll explore how to generalize this technique by creating a page layout based on a grid.

### Overview of Grid-Based Layouts

Grids are a classic layout technique that has been used in publishing for hundreds of years and, like many other publishing techniques, can be applied to web design. In a **grid layout**, the page is comprised of a system of intersecting rows and columns that form a grid. The rows are based on the page content. A long page with several articles might span several rows, or it could be a home page with introductory content that fits within a single row. The number of columns is based on the number that provides the most flexibility in laying out the page content. Many grid systems are based on 12 columns because 12 is evenly divisible by 2, 3, 4, and 6, but other sizes are also used. Figure 3–29 shows a 12-column grid layout.

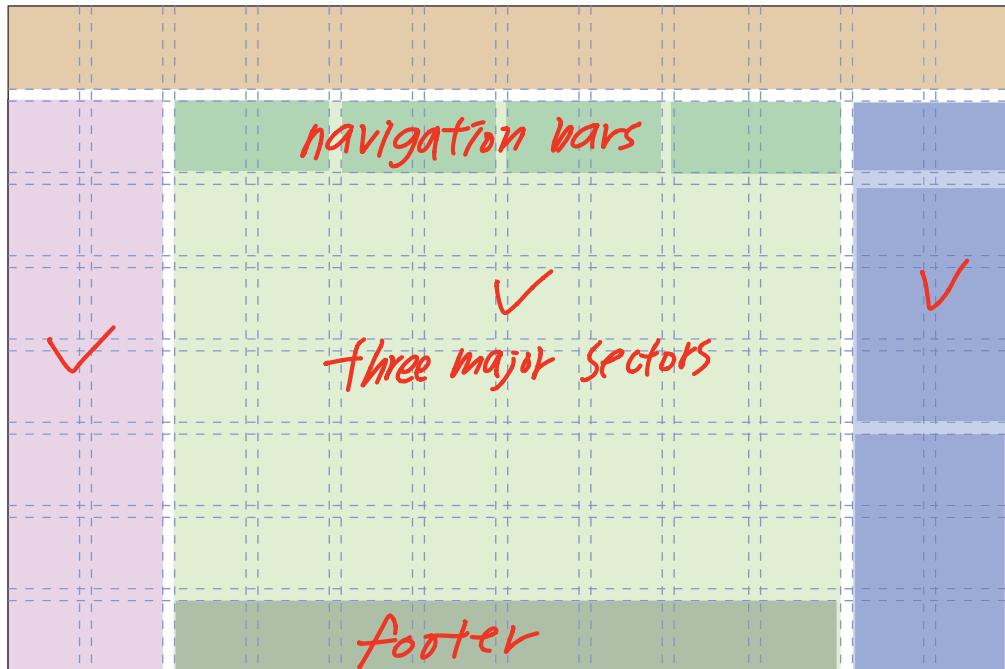
Figure 3–29 Page grid



The page designer then arranges the page elements within the chosen grid. Figure 3–30 shows one possible layout comprised of a main header element (the tan area), three major sections (the lavender, light green, and blue areas), as well as a navigation bar and a footer (the dark green areas). Some sections (like the dark green and blue areas) are further divided into small subsections.

Figure 3-30

Layout based on a grid



It should be stressed that the grid is not part of the web page content. Instead, it's a systematic approach to visualizing how to best fit content onto the page. Working from a grid has several aesthetic and practical advantages, including

- Grids add order to the presentation of page content, adding visual rhythm, which is pleasing to the eye.
- A consistent logical design gives readers the confidence to find the information they seek.
- New content can be easily placed within a grid in a way that is consistent with previously entered information.
- A well designed grid is more easily accessible for users with disabilities and special needs.
- Grids speed up the development process by establishing a systematic framework for the page layout.

There are two basic types of grid layouts: fixed grids and fluid grids.

## Fixed and Fluid Grids

960 px width

In a **fixed grid**, the widths of the columns and margins are specified in pixels, where every column has a fixed position. Many fixed grid layouts are based on a page width of 960 pixels because most desktop screen widths are at 1024 pixels (or higher) and a 960-pixel width leaves room for browser scrollbars and other features. The 960-pixel width is also easily divisible into halves, thirds, quarters, and so forth, making it easier to create evenly spaced columns.

The problem of course with a fixed grid layout is that it does not account for other screen sizes and thus, a **fluid grid**, in which column widths are expressed in percentages rather than pixels, is often used to provide more support across different devices. In the examples to follow, you'll base your layouts on a fluid grid system.

Grids are often used with responsive design in which one grid layout is used with mobile devices, another grid layout is used with tablets, and yet another layout is used with desktop computers. A layout for a mobile device is typically based on a 1-column grid, tablet layouts are based on grids of 4 to 12 columns, and desktop layouts are often based on layouts with 12 or more columns.

mobile device - 1 column  
tablet - 4 to 12 columns  
design for - 12 or more

## CSS Frameworks

Designing your own grids can be time-consuming. To simplify the process, you can choose from the many CSS frameworks available on the web. A **framework** is a software package that provides a library of tools to design your website, including style sheets for grid layouts and built-in scripts to provide support for a variety of browsers and devices.

Most frameworks include support for responsive design so that you can easily scale your website for devices ranging from mobile phones to desktop computers.

Some popular CSS frameworks include

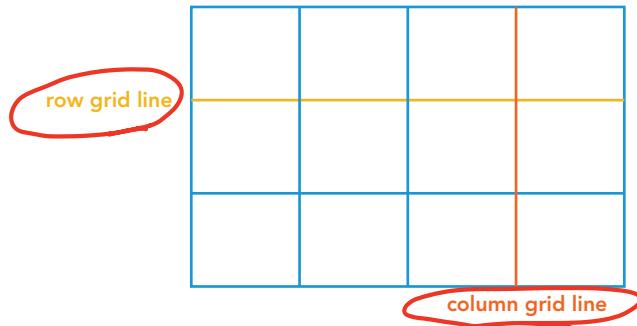
- **Bootstrap** ([getbootstrap.com](http://getbootstrap.com))
- **Neat** ([neat.bourbon.io](http://neat.bourbon.io))
- **Unsemantic** ([unsemantic.com](http://unsemantic.com))
- **Profound Grid** ([www.profoundgrid.com](http://www.profoundgrid.com))
- **HTML 5 Boilerplate** ([html5boilerplate.com](http://html5boilerplate.com))
- **Skeleton** ([getskeleton.com](http://getskeleton.com))

While a framework does a lot of the work in building the grid, you still need to understand how to interact with the underlying code, including the style sheets used to create a grid layout. In place of third-party frameworks, you can design your own grids using grid styles from CSS. Achieving Candidate Recommendation status by the W3C in December, 2017, the CSS grid styles are now widely supported by all major browsers on almost every device.

## Introducing CSS Grids

The **CSS grid model** is a set of CSS design styles used to create grid-based layouts. Before discussing the CSS styles, we should first explore the key terms and concepts associated with building a CSS grid. Each CSS grid is laid out in a set of row and column gridlines as shown in Figure 3–31.

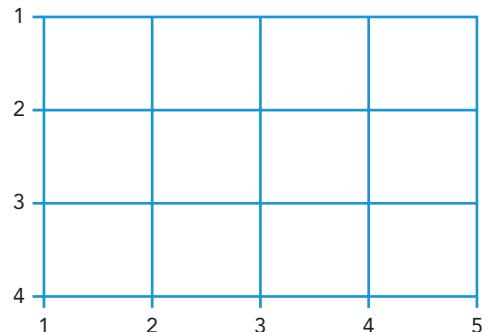
Figure 3–31 Row and column gridlines



To reference positions within a grid, the CSS grid model numbers the gridlines in the horizontal and vertical directions, starting from the top-left corner of the grid with the row gridlines and then moving left to right with the column gridlines along the bottom. Both gridlines start with a value of "1" and increase in value down and across the grid (see Figure 3–32.)

Figure 3-32

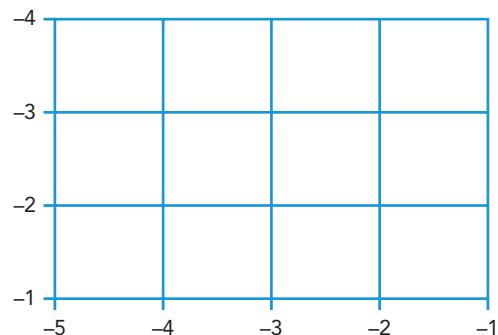
## Numbering gridlines



You can reference gridlines in the reverse order starting from the bottom-right corner with the first row and column gridlines in those directions are given a value of “-1” as shown in Figure 3-33.

Figure 3-33

## Numbering gridlines from right to left



## TIP

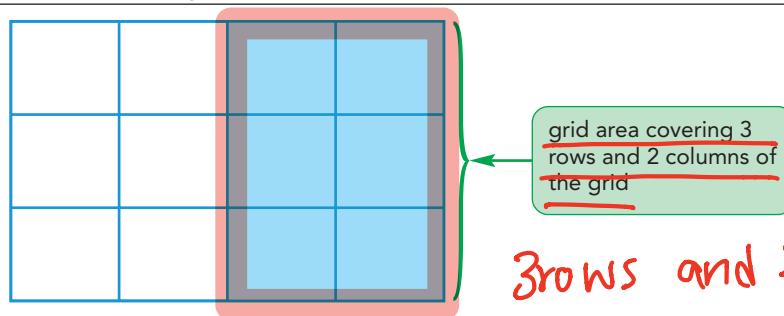
For countries and regions that read material right-to-left rather than left-to-right, the grid numbering system is reversed to reflect reading order.

The advantage of using both positive and negative gridline numbers is that you can always reference both the first gridline (1) and the last gridline (-1) no matter the size of the grid. This will become important later when placing items at specific locations within the grid or sizing those items to cover multiple rows and columns.

The cells that are created from the intersection of the horizontal and vertical gridlines will contain the elements from the web page. An element can be contained within a single cell or it can span several cells within a **grid area**. Figure 3-34 shows a grid area consisting of three rows and two columns. Note that grid areas must be rectangular; you cannot have an L-shaped grid area.

Figure 3-34

## Grid area within a CSS grid



Rows and columns are also called **tracks** or **grid tracks**.

3 rows and 2 columns of the grid

You will create a grid for a web page describing the Pandaisia Chocolates company. Anne has already written and marked up the content of the page. Open Anne's file now.

**To open the file containing information about the company:**

- 1. Use your editor to open the **pc\_about\_txt.html** file from the **html03 ▶ tutorial** folder. Enter **your name** and **the date** in the comment section and save the file as **pc\_about.html**.
- 2. Take some time to examine the contents of the page.
- 3. Open the **pc\_about.html** file in your browser. See Figure 3–35.

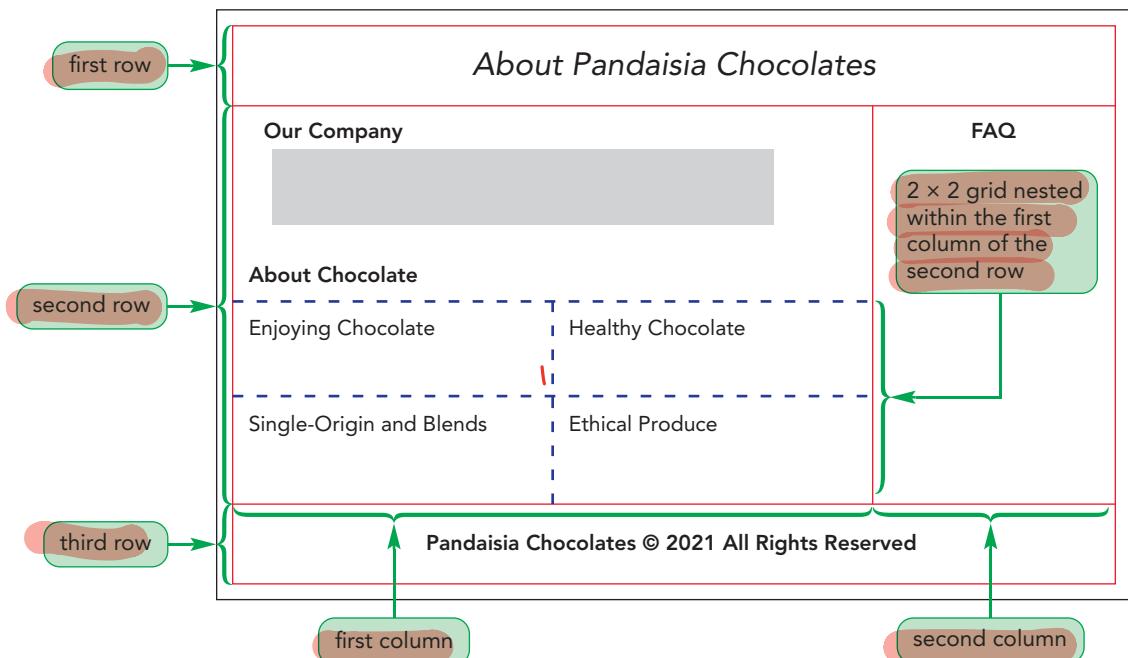
Figure 3–35

Initial About Pandaisia Chocolates page



There is currently no layout for the page contents and all the structural elements appear stacked within a single column. Anne sketches her idea for a different page layout, shown in Figure 3–36.

Figure 3–36 Proposed grid layout for the About Pandaisia Chocolates page



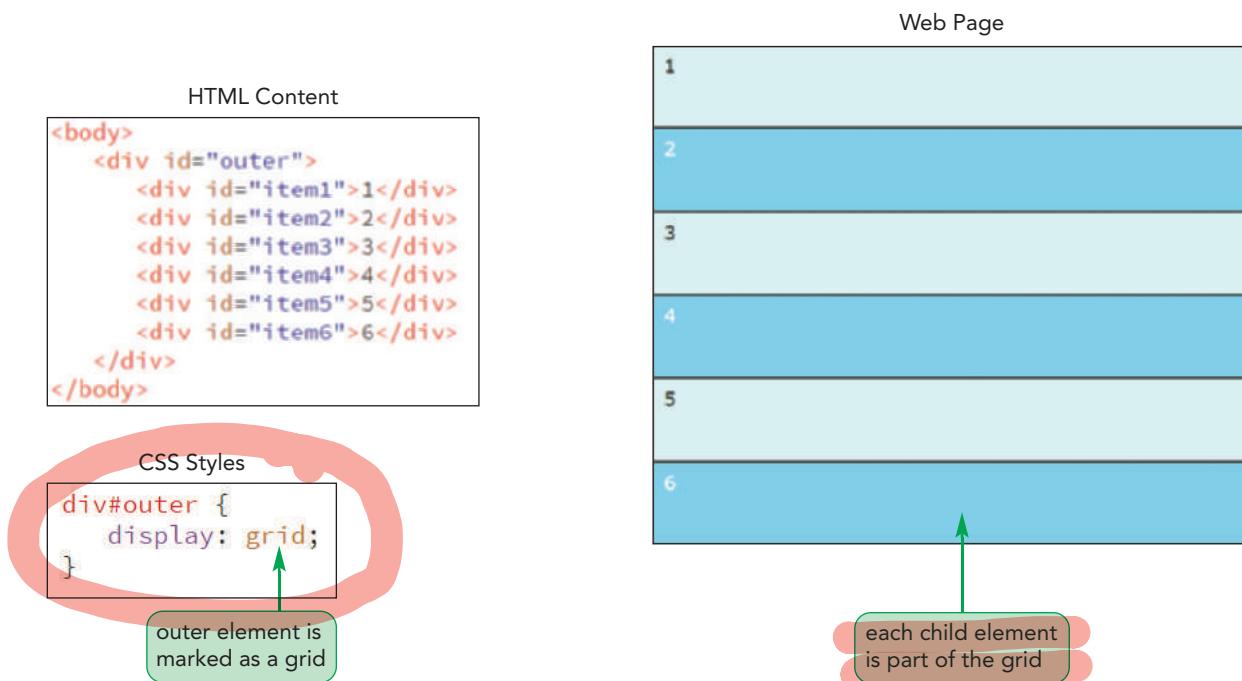
Anne's layout consists of two grids: one nested within the other. The outer grid consists of three rows and two columns. The first and third rows contain the page header and page footer, with the header and footer both spanning an entire row. The second row displays information about the company in the first column and a list of frequently asked questions in the second column. Within the second row is a nested grid of two rows and two columns containing four articles about Pandaisia Chocolates, its operations, and products. You will use the CSS grid model to create this grid layout.

## Creating a CSS Grid

To create a CSS grid, you must first identify a page element as the grid container using the following `display` property:

```
display: grid;
```

Figure 3–37 shows a simple web page containing a `div` element with the id "outer" that contains six nested `div` elements. The outer element is displayed as a grid and each of the six child elements become items within that grid. The grid is limited to those six `div` elements. Any elements nested within those `div` elements are not part of the grid structure.

Figure 3–37 Using the `display` style

The six child `div` elements are now grid items, so they are no longer considered block-level elements because they are fixed within the grid structure. You couldn't, for example, float any of those elements because floating them would remove them from the grid and the CSS grid model doesn't allow that. The entire grid itself is considered a block-level element and thus could be floated or resized within the web page just like any other block-level element.

Grids can also be created as inline elements using the style:

`display: inline-grid;`

which creates the grid inline with other elements in the web page. In this session we will only examine grids as block-level elements, but be aware than any of the techniques introduced for setting up a grid can be applied to both the block-level and inline versions.

Use the `display` property now to define outer and inner grids described in Figure 3–36. You will place all your grid styles within a separate CSS file.

### To create the grid style sheet:

- 1. Use your editor to open the `pc_grids_txt.css` file from the `html03 ▶ tutorial` folder. Enter **your name** and **the date** in the comment section and save the file as `pc_grids.css`.
- 2. Within the Grid Styles for Page Body section, insert the following style rule to define a grid for the entire page body:

```
body {
 display: grid;
}
```

- 3. Within the Grid Styles for Nested Grid section, insert the following style rule to turn the `section` element into a grid:

```
section {
 display: grid;
}
```

Figure 3–38 highlights the newly added code.

Figure 3–38

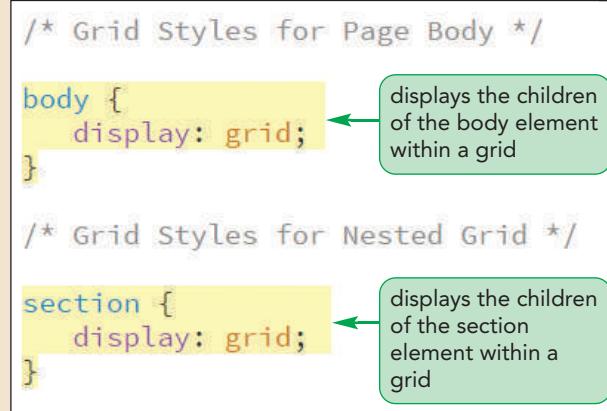
### Creating two grids for the web page

```
/* Grid Styles for Page Body */

body {
 display: grid;
}

/* Grid Styles for Nested Grid */

section {
 display: grid;
}
```



- 4. Save your changes to the file and then return to the `pc_about.html` file in your editor.
  - 5. Within the head section, add the following `link` element to link the web page to the `pc_grids.css` style sheet.
- ```
<link href="pc_grids.css" rel="stylesheet" />
```
- 6. Save your changes to the file.

Having set up the grids you will next use CSS to define the rows and columns of the grid structure.

Working with Grid Rows and Columns

To define the number and size of grid columns, use the following `grid-template-columns` style:

```
grid-template-columns: width1 width2 ...;
```

TIP

The number of columns in the grid is determined by the number of entries in the `grid-template-columns` property.

where `width1`, `width2`, etc. is a space-separated list that defines the width of the columns or tracks within the grid. For example, the following style rule creates two grid columns: the first 250 pixels in width and the second with a width of 100 pixels.

```
grid-template-columns: 250px 100px;
```

Column widths can be expressed using any CSS unit measures such as pixels, em units, and percentages. You can also use the keyword `auto` to allow the column width to be automatically set by the browser. The following style creates a three-column grid with the width of the first column fixed at 100 pixels, the third column at 50 pixels, and the center column occupying whatever space remains.

```
grid-template-columns: 100px auto 50px;
```

You might use such a layout in a page in which the first and third columns contain navigation lists that are fixed in size while the middle column contains an article that should expand to fill the remaining space.

Figure 3-39 shows a two-column grid with fixed widths of 250 pixels and 100 pixels. Notice that the number of rows is not defined, so that the browser automatically adds rows as needed to contain all page elements within the grid container. Because there are six grid items, this grid is arranged in a layout of three rows and two columns. If there were eight child elements the grid would be four rows by two columns, and so forth.

Figure 3-39 Setting the grid columns

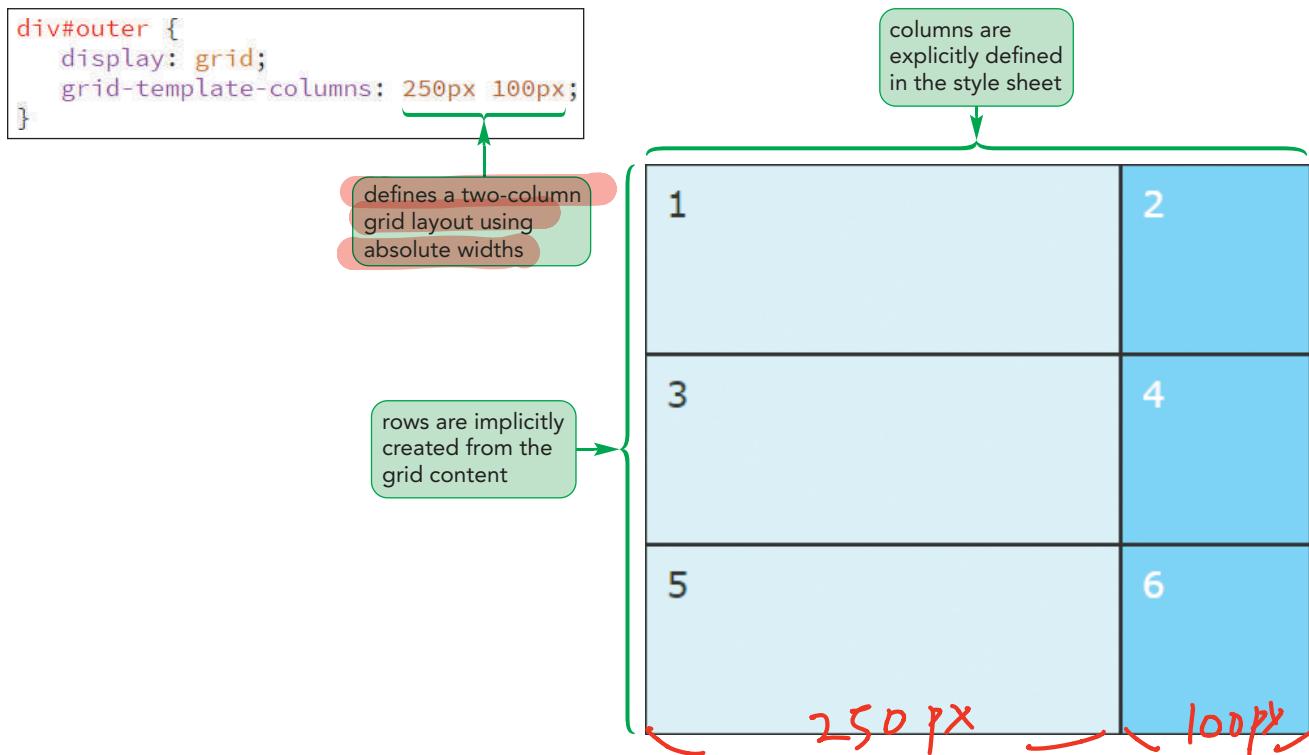


Figure 3-39 highlights an important contrast between explicit grids and implicit grids. An **explicit grid** completely defines the number and size of the grid rows and columns. An **implicit grid** contains rows and/or columns that are generated by the browser as it populates the grid with items from the grid container. In most grid layouts you will explicitly define the columns and let the browser fill out the grid rows drawn from the web page content.

To explicitly define the number of rows and their height, use the following `grid-template-rows` property:

```
grid-template-rows: height1 height2 ...;
```

where `height1`, `height2`, etc. define the heights of the grid rows. Figure 3-40 shows an example of an explicit grid in which both the columns and rows are defined in the CSS style sheet.

Figure 3-40 Explicitly defining grid columns and rows

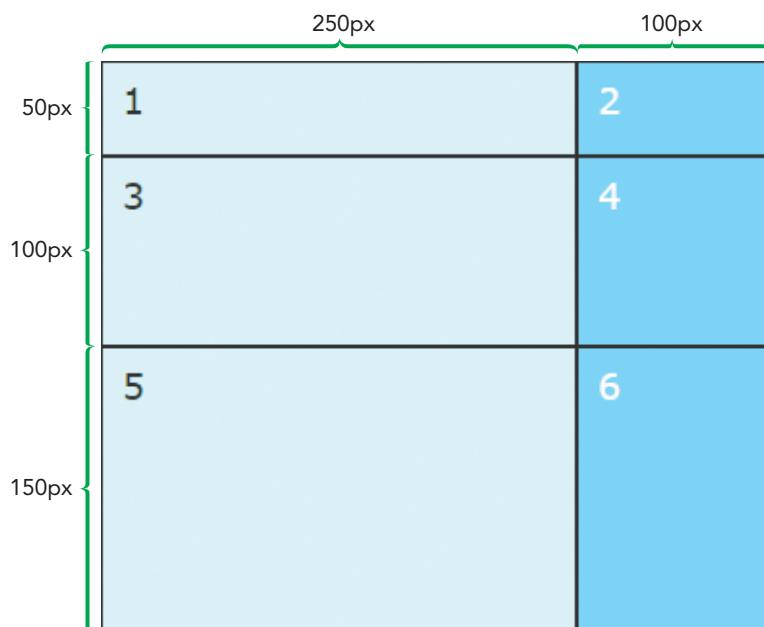
```
div#outer {
  display: grid;
  grid-template-columns: 250px 100px;
  grid-template-rows: 50px 100px 150px;
}
```

explicit grid

grid-template-columns
grid-template-rows

implicitly grid

grid-template-columns: auto
grid-auto-rows:



In an implicit grid, the row heights are determined by the page element. You can also set the row heights in an implicit grid using the following `grid-auto-rows` property:

`grid-auto-rows: height1 height2 ...;`

where `height1`, `height2`, etc. are the heights of the rows with the sequence repeating for each new set of rows. For example, the style:

`grid-auto-rows: 100px;`

sets the height of each row to 100 pixels, while the style

`grid-auto-rows: 100px 200px;`

sets the height of the first row in the implicit grid to 100 pixels, the height of the second row to 200 pixels, and then repeats those heights for each subsequent set of two rows until the grid is filled.

TIP

If the content of a grid item is greater than can be displayed within the allotted height, the browser will automatically increase the row height to match.

Track Sizes with Fractional Units

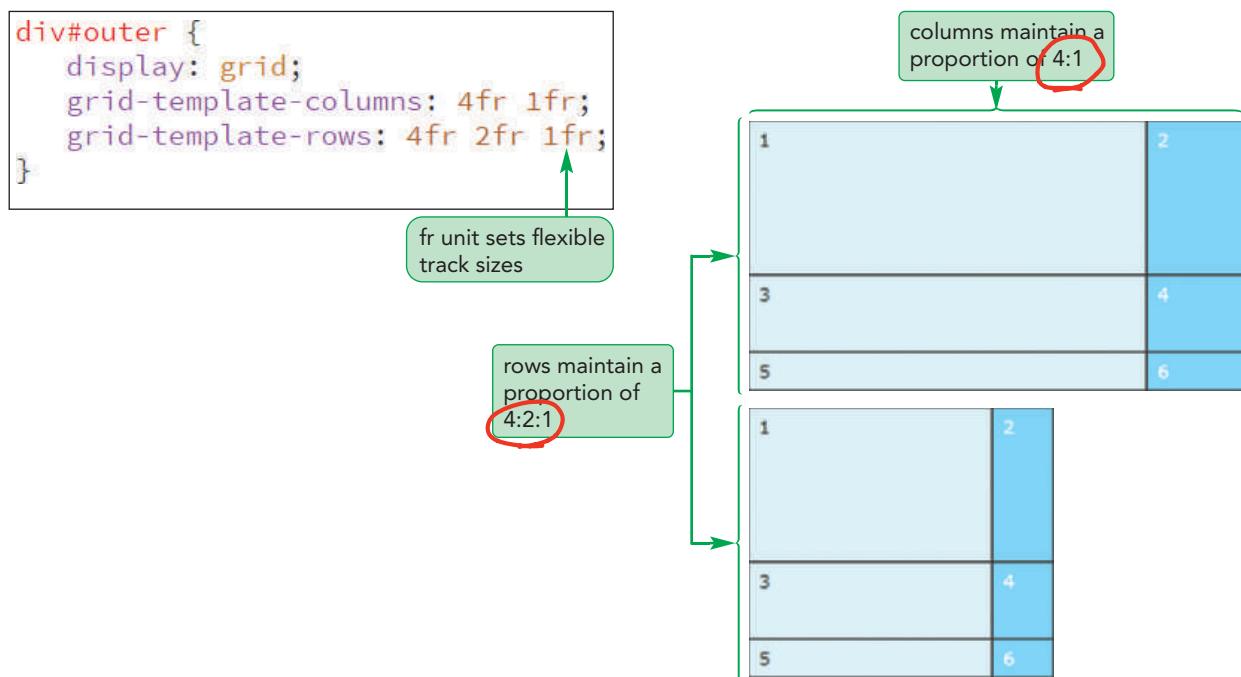
A grid layout will often need to adapt to devices of various screen widths and sizes. One way of accomplishing this is with flexible units. A **fr (fractional) unit**, indicated by the unit abbreviation `fr`, creates grid tracks that expand or contract in size to fill available space while retaining their relative proportions to one another. The following is an example of a grid in which the track sizes of the columns and rows is set using fractional units:

first column is four times wider than second

`grid-template-columns: 4fr 1fr; 4:1`
`grid-template-rows: 4fr 2fr 1fr; 4:2:1`

As the size of the display window changes, the column widths maintain their proportions so that the first column is always four times wider than the second column and the row heights maintain their proportion of 4:2:1. See Figure 3-41.

Figure 3-41 Using flexible units in a grid



Fractional units are often combined with absolute units to create grid layouts that are both fixed and flexible. The following style rule generates a grid in which the width of the first column is set to 250 pixels with the remaining space allotted to the other two columns in a proportion of 2 to 1.

grid-template-columns: 250px 2fr 1fr

Such a layout might be used in a web page in which the first column contains a navigation list whose width is fixed, the second column contains an article of primary importance, and the third column contains a sidebar of lesser importance. As the size of the display window changes, the width of the second and third columns automatically change, filling the screen while maintaining their 2:1 ratio.

Repeating Columns and Rows

Some grid layouts involve 12 or 16 columns or more. With so many columns it's difficult to specify the size of each column. You can simplify the layout style by using the following `repeat()` function

repeat(repeat, tracks)

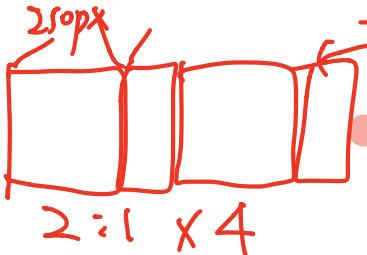
where `repeat` is the number of repetitions of the tracks specified in `tracks`. For example, the following expression creates a layout consisting of one fixed column 250 pixels in width followed by four sets of two columns in which the first column in each set is twice the width of the second column for a total of nine grid columns.

grid-template-columns: 250px repeat(4, 2fr 1fr);

In place of a `repeat` value, you can use the keyword `auto-fill` to fill up the grid with as many columns (or rows) that will fit within the grid container. The following style uses the `auto-fill` keyword to fill the grid with as many 100 pixel-wide columns that will fit within the container:

grid-template-columns: 250px repeat(auto-fill, 100px);

Any grid item that cannot fit within the grid container will be automatically wrapped to a new row. This type of layout could be used with an image gallery in which each



row contains as many 100 pixel-wide images that can fit within the display window, arranged in columns.

Finally, you can switch between fixed and flexible track sizes using the following `minmax()` function

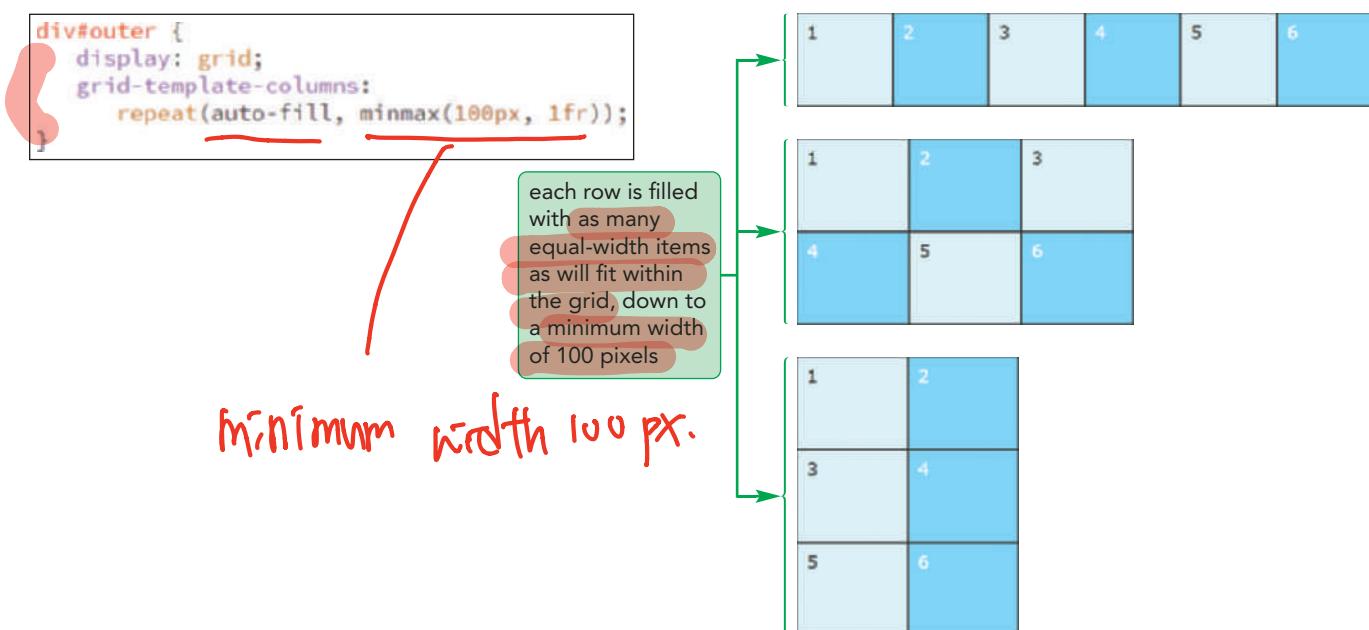
`minmax(min, max)`

where `min` is the minimum track size for a row and column and `max` is the maximum. Used in the following style rule, the grid will contain as many columns of equal width that can fit within a grid container down to a minimum width of 100 pixels:

`grid-template-columns: repeat(auto-fill, minmax(100px, 1fr));`

Figure 3–42 shows how such a layout would be applied to grid containers of different widths.

Figure 3–42 Using the `minmax` function in a grid



As the grid container decreases in size, grid items are automatically wrapped to a new row. Under each layout, the columns are given equal widths down to a minimum width value of 100 pixels.

Applying a Grid Layout

Now that you've seen how to set the size of rows and columns within a grid, you will apply your knowledge to the About Pandaisia web page. From Anne's proposed layout shown earlier in Figure 3–36, you've learned that the two columns in the outer grid should be displayed in a proportion of 2:1, while the four articles about chocolate in the nested grid should be displayed with columns of equal width. You will define the column widths for both the outer and nested grids using fractional units. You won't, however, explicitly define the number and height of the grid rows, leaving the browser to implicitly lay out that content.

To define the grid columns:

- 1. Return to the `pc_grids.css` file in your editor.
- 2. Within the style rule for the `body` element, add the style:

`grid-template-columns: 2fr 1fr;`

- 3. Within the style rule for the section element, add:

```
grid-template-columns: repeat(2, 1fr);
```

Figure 3–43 highlights the newly added code.

Figure 3–43 Creating two grids for the web page

```
/* Grid Styles for Page Body */

body {
  display: grid;
  grid-template-columns: 2fr 1fr; ← creates two grid columns with the first column twice the width of the second
}

/* Grid Styles for Nested Grid */

section {
  display: grid;
  grid-template-columns: repeat(2, 1fr); ← creates two grid columns of equal width
}
```

- 4. Save your changes to the file and then reload pc_about.html in your browser. Figure 3–44 shows part of the layout of the page under the current grid structure.

Figure 3–44 Web page with the column layout



The page layout does not resemble the plan that Anne outlined in Figure 3–36 because we have not specified where each item should be placed within the grid. That will be the final step in designing the grid layout. Before doing that however, it would be helpful to view the page with gridlines superimposed on the web page. We can do that with outline styles.

Outlining a Grid

Outlines are simply lines drawn around an element, enclosing the element content, padding, and border spaces. Unlike borders, which you'll study in the next tutorial, an outline doesn't add anything to the width or height of the object, it only indicates the extent of the element on the rendered page.

The width of the line used in the outline is defined by the following `outline-width` property

`outline-width: value;`

where `value` is expressed in one of the CSS units of length, or with the keywords `thin`, `medium`, or `thick`. The line color is set using the `outline-color` property

`outline-color: color;`

where `color` is a CSS color name or value. Finally, the design of the line can be set using the following `outline-style` property

`outline-style: style;`

where `style` is `none` (to display no outline), `solid` (for a single line), `double`, `dotted`, `dashed`, `groove`, `inset`, `ridge`, or `outset`. All the outline properties can be combined into the following `outline` shorthand property

`outline: width style color;`

where `width`, `style`, and `color` are the values for the line's width, design, and color. For example, the following style rule uses the wildcard selector along with the `outline` shorthand property to draw a 1 pixel dotted green line around every element on the web page:

```
* {
    outline: 1px dotted green;
}
```

TIP

Outlines can also be applied to inline elements such as inline images, citations, quotations, and italicized text.

Note that there are no separate outline styles for the left, right, top, or bottom edge of the object. The outline always surrounds an entire element.

REFERENCE

Adding an Outline to an Element

- To add an outline around an element, use the property

`outline: width style color;`

where `width`, `style`, and `color` are the outline width, outline design, and outline color respectively. These attributes can be listed in any order.

Use the `outline` property now to add an outline to each item in both the outer and inner grids.

TIP

Most browsers include developer tools for viewing the gridlines from a CSS grid. See your browser documentation for specific instructions.

To define the grid columns:

1. Return to the `pc_grids.css` file in your editor.
2. At the bottom of the style sheet, add the following style rule to place a red dashed outline around every child of the `body` element:

```
body > * {  
    outline: 2px dashed red;  
}
```

3. Add the following style rule to place a blue dashed outline around every child of the `section` element:

```
section > * {  
    outline: 2px dashed blue;  
}
```

Figure 3–45 highlights the code for the style rules.

Figure 3–45**Adding outlines to grid items**

```
section {  
    display: grid;  
    grid-template-columns: repeat(2, 1fr);  
}  
  
body > * {  
    outline: 2px dashed red;  
}  
  
section > * {  
    outline: 2px dashed blue;  
}
```

4. Save your changes to the file and then reload `pc_about.html` in your browser. Figure 3–46 shows the outlines around both the outer and inner grids.

Figure 3–46

Web page with grid items outlined



Adding an outline makes it clear how each item is placed within the grid. Next you will change the location and sizes of the grid items to match Anne's proposed layout.

Placing Items within a Grid

By default, grid items are laid out in document order going from left to right and up to down, with each item placed within a single cell. Thus, the page header in Figure 3–46, being the first item in the grid, appears in the first row and column. The next item, an article about the company, occupies the cell in the second column of the first row. Subsequent items are placed in cells in the next rows filling the grid until the last item is reached, which in this case is the page footer. In many layouts however, you might want to move items around or have a single item occupy multiple rows and columns.

Defining Grids with CSS

- To assign a CSS grid to an element, use the property
`display: grid;`

`grid-template-rows: height1 height2 ...;`
`grid-template-columns: width1 width2 ...;`

where `height1`, `height2`, `width1`, `width2`, etc. define the row heights or column widths.

- To place an element within a specific intersection of grid rows and columns, use the properties

`grid-row-start: integer;`
`grid-row-end: integer;`
`grid-column-start: integer;`
`grid-column-end: integer;`

where `integer` defines the starting and ending row or column that contains the content.

- To more compactly set the location of the element within the grid, use the properties

`grid-row: start/end;`
`grid-column: start/end;`

where `start` and `end` are the starting and ending coordinates of the row and columns containing the element.

Placing Items by Row and Column

To move a grid item to a specific location within the grid, use the following `grid-row` and `grid-column` properties:

`grid-row: row;`
`grid-column: column;`

where `row` is the row number and `column` is the column number. Thus, to place the `article` element in a **grid cell** located in the first row and second column of the grid, apply the following style rule:

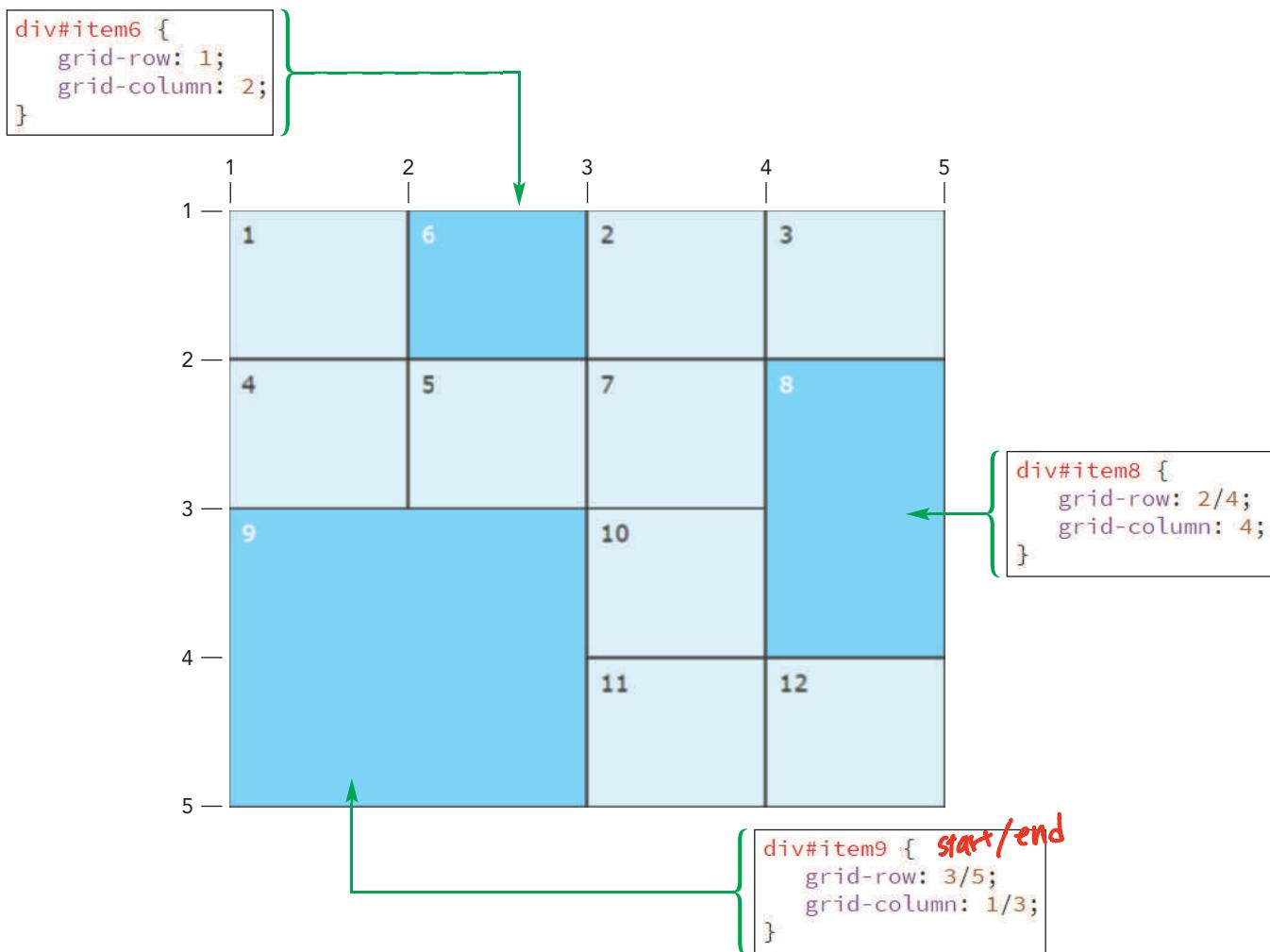
```
article {  
    grid-row: 1;  
    grid-column: 2;  
}
```

To extend a grid item so that it covers multiple rows or multiple columns, include the starting and ending gridline in the style property as follows:

`grid-row: start/end;`
`grid-column: start/end;`

where `start` is the starting gridline and `end` is the ending gridline. Figure 3–47 shows a page layout in which grid items 6, 8, and 9 have been moved and resized using the `grid-row` and `grid-column` properties. For example, item 6 is moved to the first row and second column of the grid while items 8 and 9 have been resized to cover multiple rows and/or columns. The other items in the grid are placed in their default locations and sized to fit within a single grid cell.

Figure 3-47 Placing items within a grid layout



Starting and ending gridlines can also be expressed in the following four properties:

grid-column-start: *integer*;
 grid-column-end: *integer*;
 grid-row-start: *integer*;
 grid-row-end: *integer*;

so that the style rule grid-column: 2/5 is equivalent to:

grid-column-start: 2;
 grid-column-end: 5;

Which approach you use is a matter of personal preference.

You can also use negative gridline numbers (shown earlier in Figure 3-33) to extend an item from the first gridline to the last. Recall that since the last column or row gridline has a value of -1, the expression

grid-column: 1/-1;

would extend the grid item across the entire row from the first gridline to the last. Similarly, the expression

grid-row: 1/-1;

would create a grid item that extends across an entire column. Note that this technique only works with explicit grids because in an implicit grid there is no defined last column or row.

INSIGHT

Naming Gridlines

Gridline numbers can be difficult and cumbersome to work with, so the CSS grid model also supports **gridline names**, which are descriptive names for row and column gridlines. Gridline names are created by adding a name enclosed within square brackets into the `grid-template-columns` or `grid-template-rows` style. For example, the following style creates a grid with three columns and four column gridlines named `row-start`, `main-start`, `main-end`, and `row-end`.

```
grid-template-columns: [row-start] 50px [main-start] 250px  
[main-end] 100px [row-end];
```

To extend a grid item across the entire row, you could apply the style:

```
grid-column: 1/4;
```

or

```
grid-column: row-start/row-end;
```

An article could be placed within the center grid column with the style:

```
grid-column: main-start/main-end;
```

Gridline names can make your CSS code easier to interpret and manage and can be more easily updated if you insert additional rows or columns within your grid layout.

Using the span Keyword

Another way of setting the size of a grid cell is with the **span keyword**. The general syntax is:

```
grid-row: span value;  
grid-column: span value;
```

where `value` is the number of rows or columns covered by the item. The following style rule extends the `article` element across 2 rows and 3 columns of the grid.

```
article {  
    grid-row: span 2;  
    grid-column: span 3;  
}
```

To specify both the location and the size of the item, include the starting gridline in the style rule. The following style rule places the `article` element in the first row and fourth column of the grid while spanning two rows and three columns from that location.

```
article {  
    grid-row: 1/span 2;  
    grid-column: 4/span 3;  
}
```

In Anne's proposed layout, the page header should occupy a single row, extending from the first column to the last. Use the `grid-column` style rule now to display the body header across the first row of the grid.

To place the body header across the first row:

- 1. Return to the **pc_grids.css** file in your editor.
- 2. Directly below the style rule for the body element, insert the following style rule as shown in Figure 3-48.

```
body > header {
  grid-row: 1;
  grid-column: 1/-1;
}
```

Figure 3-48

Spanning the body header across the grid row

```
body {
  display: grid;
  grid-template-columns: 2fr 1fr;
}

body > header {
  grid-row: 1;
  grid-column: 1/-1;
}
```

header placed in the first grid row

body header extends from the first gridline to the last

number of the first gridline

number of the last gridline

- 3. Save your changes to the file and reload **pc_about.html** in your browser. The body header extends across the first row of the grid (see Figure 3-49).

Figure 3-49

Body header in the first row

body header occupies the first row

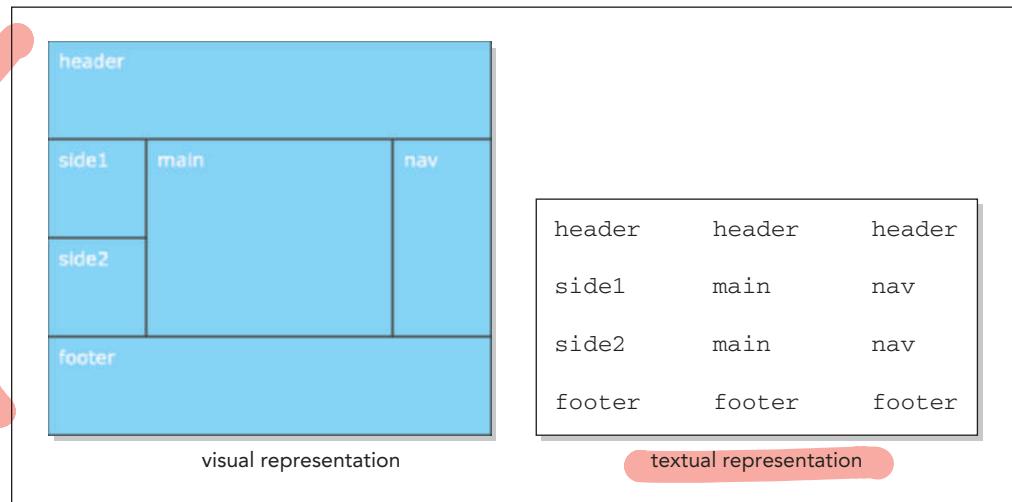
Gridlines are a quick and effective method of placing and sizing grid items, but they can be confusing when applied to a grid of several rows and columns. An easier and more intuitive approach is to use grid areas.

Placing Grid Items by Area

In the grid areas approach to layout you identify sections of the grid with item names, creating a textual representation of the layout. Figure 3–50 shows a grid of four rows and three columns in which several items span multiple rows and columns, represented both visually as it would appear on the web page and textually.

Figure 3–50

Mapping out grid areas



To create a textual representation in a style sheet, use the following `grid-template-areas` property:

```
grid-template-areas: "row1"
                    "row2"
                    ...;
```

where `row1`, `row2`, etc. are text strings containing the names of the areas for each row. Thus, to create the grid layout shown in Figure 3–50, you would enter the style:

```
grid-template-areas: "header header header"
                    "side1 main nav"
                    "side2 main nav"
                    "footer footer footer";
```

You will add a `grid-template-areas` property to the style sheet, representing the layout Anne proposed in Figure 3–36.

To place the body header across the first row:

- 1. Return to the `pc_grids.css` file in your editor.
- 2. Within the style rule for the body element, insert the following style:

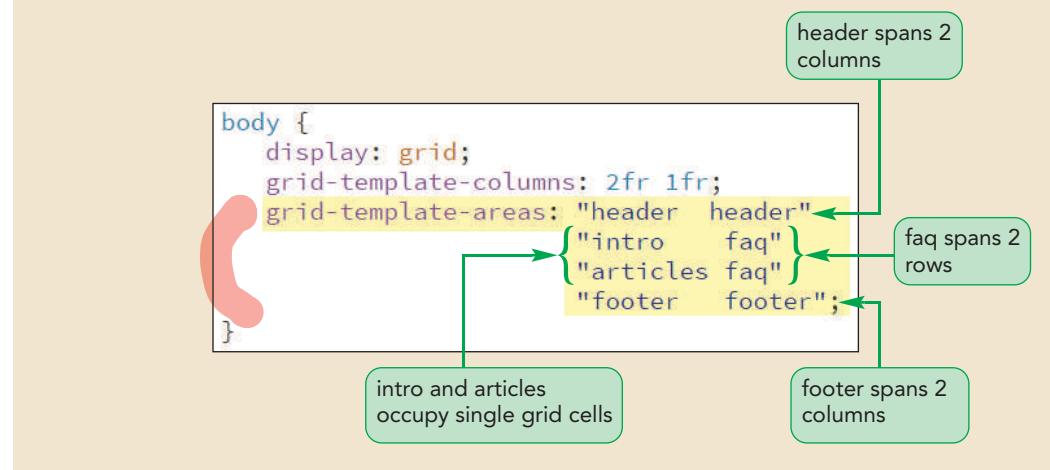
```
grid-template-areas: "header    header"
                    "intro     faq"
                    "articles  faq"
                    "footer    footer";
```

See Figure 3–51.

Make sure you enclose each row of the grid layout within a set of quotation marks.

Figure 3–51

Defining areas for the outer grid



To assign elements to grid areas, use the following `grid-area` property:

```
grid-area: area;
```

where `area` is the name of an area defined in the `grid-template-areas` property. Use the `grid-area` property now to assign elements from the web page to areas within the grid.

To assign the page elements to grid areas:

- 1. Below the `body > header` style rule, add the following style to assign the `article` element to the `intro` grid area.


```
body > article {grid-area: intro;}
```
- 2. Place the `aside` element in the `faq` grid area with the style:


```
body > aside {grid-area: faq;}
```
- 3. Place the `section` element in the `articles` grid area with the style:

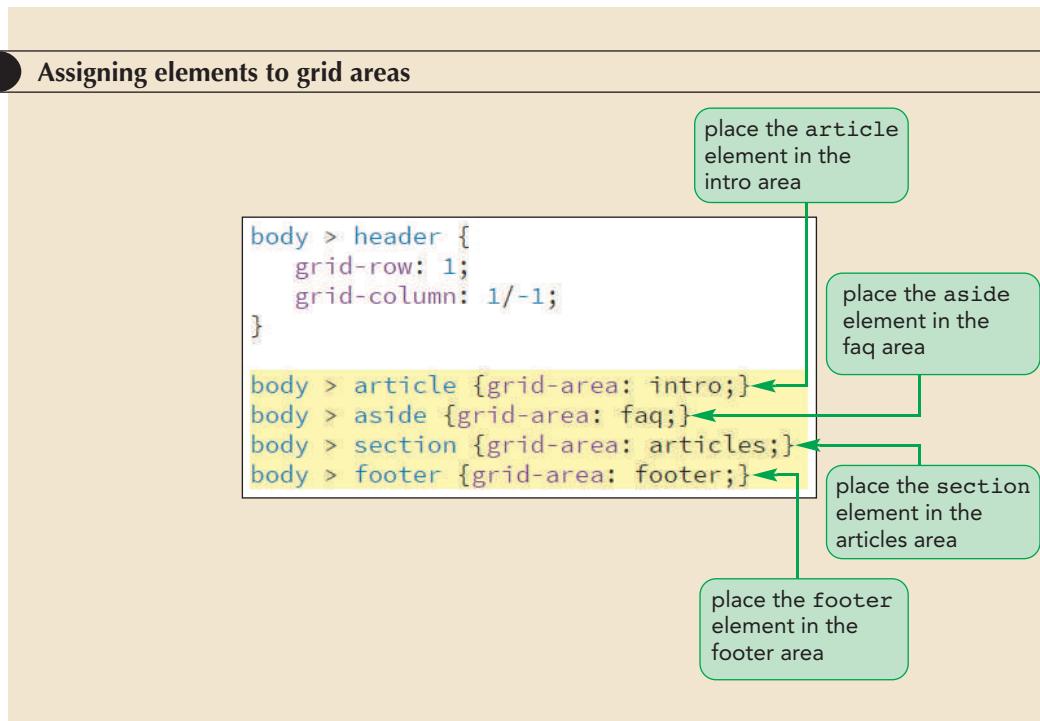

```
body > section {grid-area: articles;}
```
- 4. Place the `body footer` element in the `footer` grid area using the style:


```
body > footer {grid-area: footer;}
```

Figure 3–52 highlights the newly added code.

Figure 5–52

Assigning elements to grid areas



The `grid-area` property can also be used as a shorthand to place and size grid items using gridline numbers. The general syntax is:

`grid-area: row-start/col-start/row-end/col-end;`

where `row-start`, `col-start`, `row-end`, and `col-end` are the starting and ending gridline numbers from the grid's rows and columns. For example, the following expression places the grid item to extend from the first row gridline through the fourth and from the third column gridline through the fifth.

`grid-area: 1/3/4/5;`

The only remaining part of the About Pandaisia web page that needs to be placed within the layout is the “About Chocolate” `h1` heading that appears in the nested grid. Anne wants this heading to extend across two columns in the first row of that grid. Add a style rule to place the heading now.

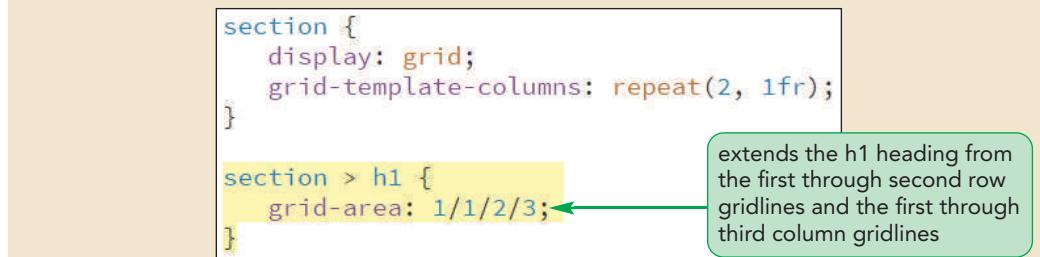
To place the `h1` heading:

- 1. Below the style rule for the `section` element, add the following rule to place the `h1` heading:

```
section > h1 {
  grid-area: 1/1/2/3;
}
```

See Figure 3–53.

Figure 3–53

Placing the `h1` heading

2. Save your changes to the `pc_grids.css` file and then reload the `pc_about.html` file in your browser. Figure 3–54 shows the complete layout of the page.

Figure 3–54

About Pandaisia web page

The screenshot shows a website for Pandaisia Chocolates. At the top, there is a header with the company name in a large, stylized font. Below the header, a sub-header reads "About Pandaisia Chocolates". The main content area is organized into a grid layout. The first column contains sections for "Our Company" (with a photo of a chocolate-making process) and "About Chocolate". The second column contains sections for "Enjoying Chocolate" and "Healthy Chocolate". The third column contains sections for "Single-Origin and Blends" and "Ethical Produce". A sidebar on the right is titled "FAQ" and contains several questions and answers about chocolate products and availability. The footer at the bottom of the page includes a copyright notice: "Pandaisia Chocolates © 2021 All Rights Reserved".

3. Return to the `pc_grids.css` file in your text editor.
4. Remove the two style rules that create the red and blue dashed outlines and then save your changes to the file.
5. Reload the `pc_about.html` file in your browser and confirm that the grid outlines are removed from the rendered page.

Compare the appearance of the page content in Figure 3–54 with the schematic diagram shown earlier in Figure 3–36 to see how using a grid provided a unified layout for the page. As you become more experienced with setting up and applying grids, you can move to more intricate and dynamic page layouts.

INSIGHT

Generating Content with *Placeholder Text*

Placeholder text, also known as *lorem ipsum*, is nonsensical, improper Latin commonly used in page design as filler text. Rather than creating large portions of sample text before you can view your layout, placeholder text is used to quickly generate sentences, lines, and paragraphs that resemble the structure and appearance of real text. Placeholder text is a particularly useful tool for web designers because they can begin working on page design without waiting for their clients to supply all the page content.

That previous paragraph is an example of **placeholder text**, which is nonsensical, improper Latin commonly used in page design as filler text. Rather than creating large portions of sample text before you can view your layout, placeholder text is used to quickly generate sentences, lines, and paragraphs that resemble the structure and appearance of real text. Placeholder text is a particularly useful tool for web designers because they can begin working on page design without waiting for their clients to supply all the page content.

Many popular web editors include tools to generate placeholder text strings in a wide variety of formats and styles. There are also placeholder text generators freely available on the web that supplement the placeholder text with HTML markup tags.

Defining the Grid Gap

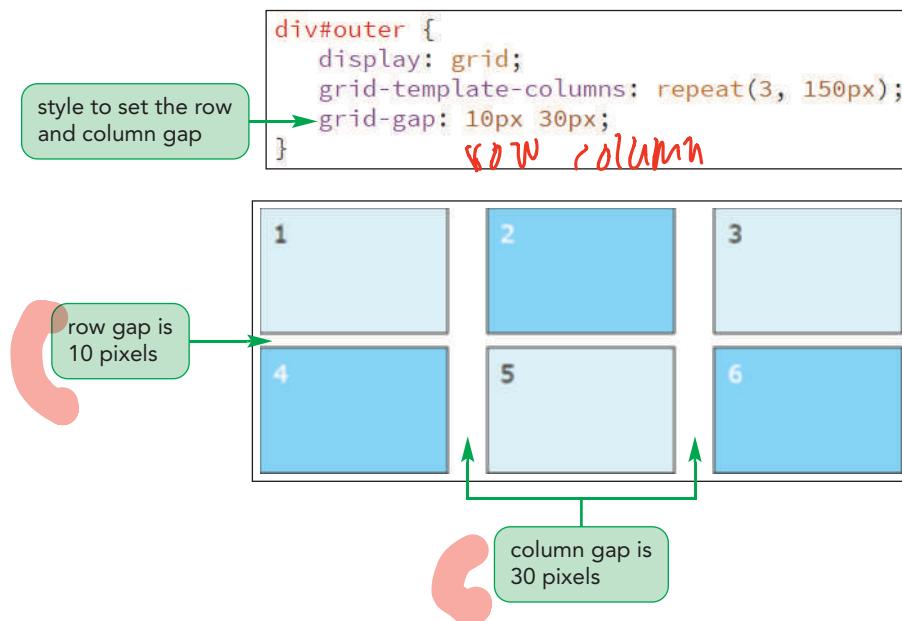
Another part of grid layout is defining the space between items in a grid. So far, all our layouts have assumed no spacing, but many layouts include interior spaces to allow each item “room to breathe.” The gap size is defined using the following `grid-gap` property:

```
grid-gap: row column;
```

where `row` is the internal space between grid rows and `column` is the internal space between grid columns. Figure 3–55 shows a grid layout in which the rows are separated by a 10-pixel space and the columns by a space of 30 pixels.

Figure 3–55

Setting the grid gap



You can also set the grid gaps for rows and columns using the following properties:

```
grid-column-gap: value;
grid-row-gap: value;
```

where *value* is the size of the gap in one of the CSS units of measure. Anne wants you to add a 15-pixel column gap to the About Pandaisia web page but leave the row gap at its default value of 0 pixels.

To set the size of the column gap:

- 1. Return to the **pc_grids.css** file in your editor.
- 2. Within the style rule for the body element, add the following property to set the column gap size as shown in Figure 3–56.

```
grid-column-gap: 15px;
```

Figure 3–56

Setting the size of the column gap

```
body {
  display: grid;
  grid-template-columns: 2fr 1fr;
  grid-template-areas: "header header"
                      "intro  faq"
                      "articles  faq"
                      "footer  footer";
  grid-column-gap: 15px;
}
```

interior space between columns set to 15 pixels

- 3. Save your changes to the file then reload the **pc_about.html** file in your browser. As shown in Figure 3–57, the gap between the first and second columns is set to 15 pixels.

Figure 5–57

Gap between the first and second columns



- 4. You've completed your work on the web page. Close the **pc_about.html** and **pc_grids.css** files.

Note that, unlike margins, the gap size setting is applied only to the interior space between the grid items and not to the exterior space between the grid items and the edge of the grid container.

Managing Space within a Grid

The grid-gap property allows you to define the space between grid items. CSS includes other properties to manage the space around the content of each grid cell. By default, there is no space between the grid cell borders and the grid cell content. However, you can position the content within the grid cell using the align-items and justify-items properties. The align-items property sets the vertical placement of the content, while the justify-items property sets the horizontal placement. The syntax of both properties is:

```
align-items: placement;
justify-items: placement;
```

where *placement* is:

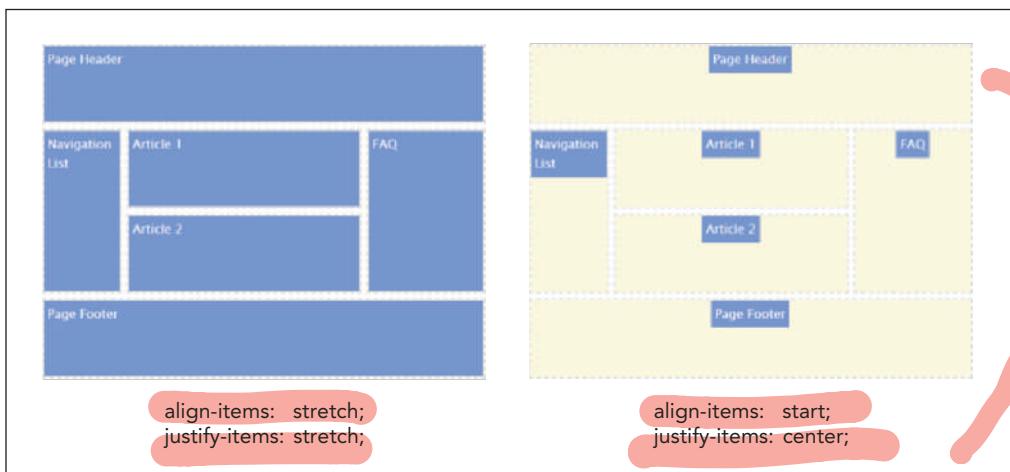
- stretch to expand the content between the top/bottom or left/right edges, removing any spacing between the content and the cell border (the default)
- start to position the content with the top or left edge of the cell
- end to position the content with the bottom or right edge of the cell
- center to center the content vertically or horizontally within the cell

TRY IT
You can explore the align-items and justify-items properties using the demo_grid1.html file from the html03 ▶ demo folder.

Figure 3–58 shows the impact of the align-items and justify-items properties on the placement of the content within each grid cell. By default, there is no spacing between the content and the cell border as the content “stretches” to fill the cell (shown in the grid on the left in the figure). In the grid on the right, the content is placed at the start (top) and horizontal center of the cell and spacing is added between the cell content and the cell borders.

Figure 3–58

Applying the align-items and justify-items properties



Alignment for a Single Grid Cell

The `align-items` and `justify-items` properties apply to every cell in the grid. To align and justify only one cell, apply the `align-self` and `justify-self` properties to the content within the grid cell. The placement values are the same as for the `align-items` and `justify-items` properties. For example, the following style rule displays the content of the `article` element in both the horizontal and vertical center of the grid cell.

```
C article {  
    align-self: center;  
    justify-self: center;  
}
```

Using the `align-self` and `justify-self` properties is a quick and easy way of centering your content within the web page. Before the introduction of the CSS grid model, this was a difficult task involving a lot of CSS hacks, but now it can be accomplished by simply placing the item within a grid and centering the content both horizontally and vertically.

Aligning the Grid

In some layouts involving fixed units, the space taken up by the items within the grid will be less than the total space allotted to the grid container itself, creating unused space in the container. By default, the grid will be positioned at the top-left corner of the container, but you can modify that position using the `align-content` and `justify-content` properties:

```
C align-content: placement;  
justify-content: placement;
```

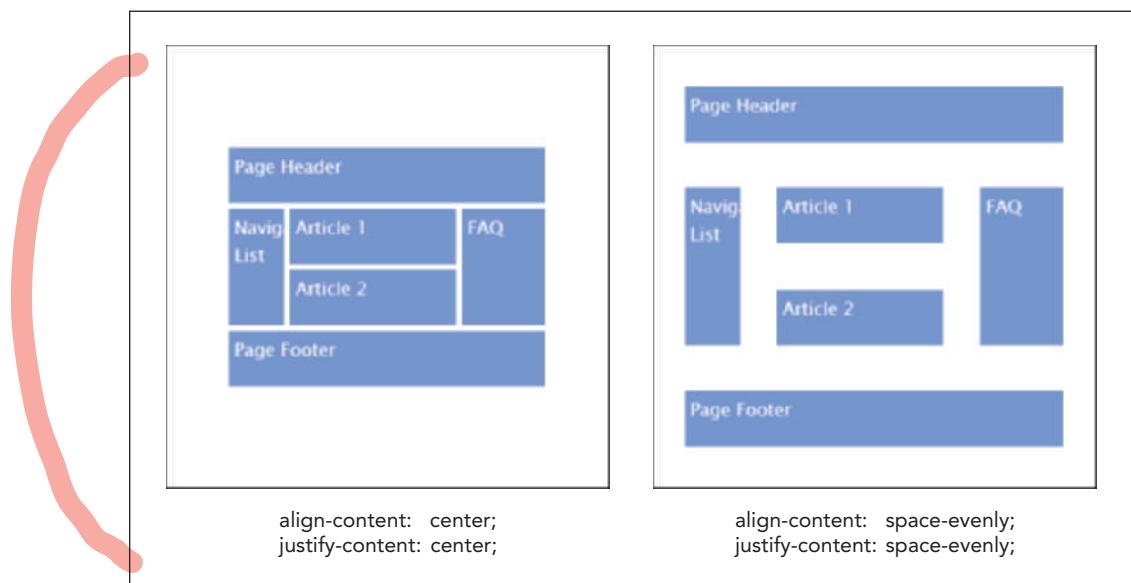
where `placement` is:

- `start` to position the grid with the top or left edge of the container (the default)
- `end` to position the grid with the bottom or right edge of the container
- `center` to center the grid vertically or horizontally within the container
- `space-around` to insert an even amount of space between each grid item, with half-sized spaces on the far ends
- `space-between` to insert an even amount of space between each grid item, with no space at the far ends
- `space-evenly` to insert an even amount of space between each grid item, including the far ends

As with the other grid properties, the `align-content` property positions the grid vertically within the container and the `justify-content` property moves the grid horizontally. Figure 3-59 shows how the interior space within the grid container can be managed using the `align-content` and `justify-content` properties. In the left grid, the layout is centered both horizontally and vertically within the container. In the right grid, the grid items themselves are positioned evenly within the container.

You can explore the `align-content` and `justify-content` properties using the `demo_grid2.html` file from the `html03 ▶ demo` folder.

Figure 3–59 Applying the align-content and justify-content properties



The `align-content` and `justify-content` properties are useful when you want to center your entire layout within the web page in both the horizontal and vertical directions.



PROSKILLS

Written Communication: Getting to the Point with Layout

Page layout is one of the most important aspects of web design. A well-constructed layout naturally guides a reader's eyes to the most important information in the page. You should use the following principles to help your readers quickly get to the point:

- **Guide the eye.** Usability studies have shown that a reader's eye first lands in the top center of the page, then scans to the left, and then to the right and down. Arrange your page content so that the most important items are the first items a user sees.
- **Avoid clutter.** If a graphic or an icon is not conveying information or making the content easier to read, remove it.
- **Avoid overcrowding.** Focus on a few key items that will be easy for readers to locate while scanning the page, and separate these key areas from one another with ample white space. Don't be afraid to move a topic to a different page if it makes the current page easier to understand.
- **Make it manageable.** It's easier for the brain to process information when it's presented in smaller chunks. Break up long extended paragraphs into smaller paragraphs or bulleted lists.
- **Use a grid.** Users find it easier to digest content when it's aligned vertically and horizontally. Using a grid helps you line up your elements in a clear and consistent way.
- **Cut down on distractions.** If you're thinking about using blinking text or a cute animated icon, don't. The novelty of such features wears off very quickly and distracts users from the valuable content.

Always remember that your goal is to convey information to readers, and that an important tool in achieving that is to make it as easy as possible for readers to find that information. A thoughtfully constructed layout is a great aid to effective communication.

In the next session, you'll explore CSS positioning styles that allow you to place page elements anywhere within the rendered page.

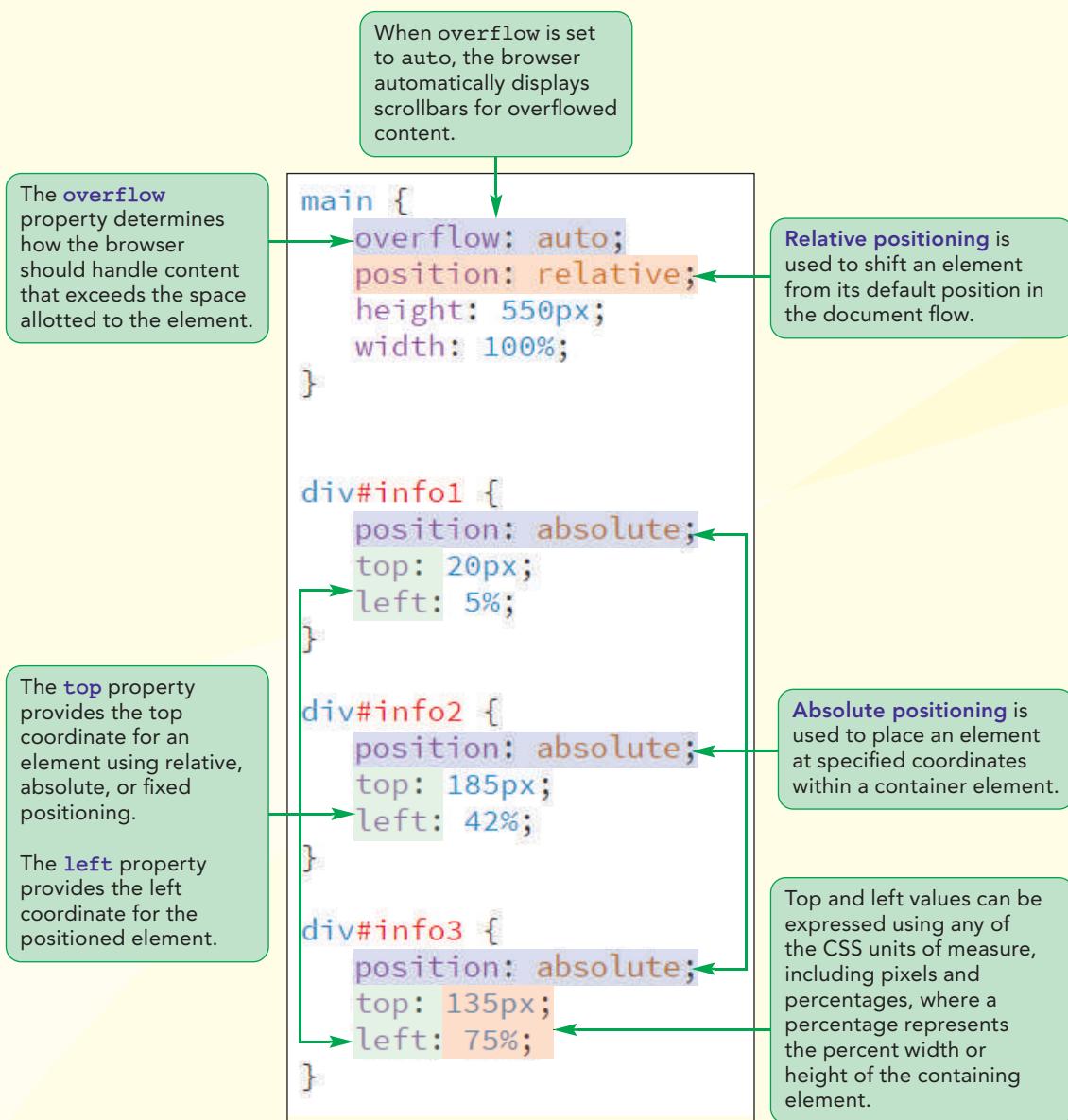
REVIEW

Session 3.2 Quick Check

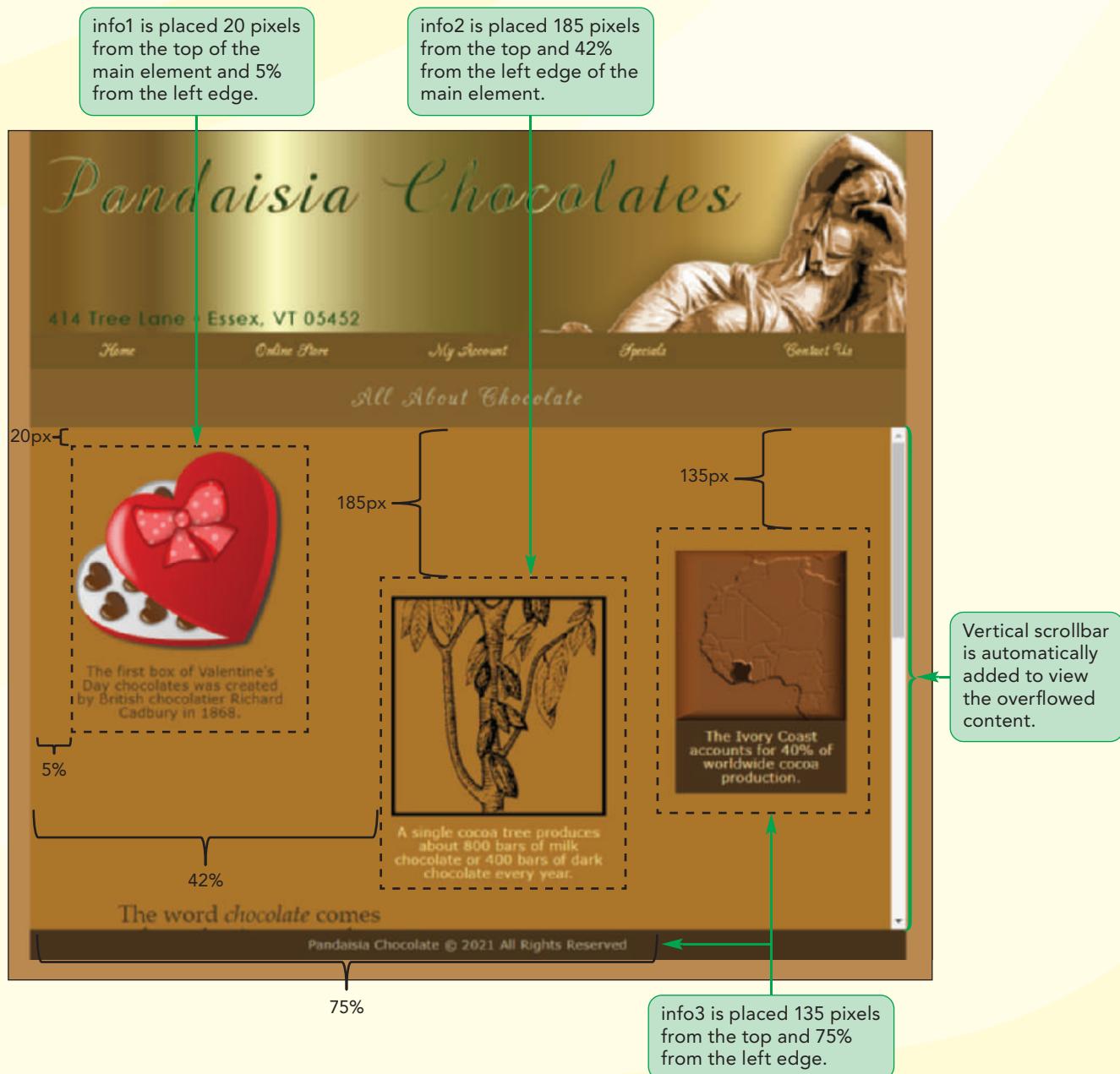
1. To change an element into a grid container, apply the style:
 - a. `display: grid-container;`
 - b. `grid-template-columns: auto;`
 - c. `display: grid;`
 - d. All of the above
2. A fractional unit (`fr`) is:
 - a. part of a pixel
 - b. a fraction of an em unit
 - c. a fraction of any fixed unit
 - d. used to create elements that expand or contract to fill available space
3. To explicitly define the columns within a grid, use the CSS property:
 - a. `grid-template-columns`
 - b. `grid-columns`
 - c. `columns`
 - d. `grid-auto-columns`
4. To implicitly define the height of grid rows, use:
 - a. `grid-template-rows`
 - b. `grid-rows`
 - c. `rows`
 - d. `grid-auto-rows`
5. To position a grid item in the second row and cover the second and third column, apply the style(s):
 - a. `grid-row: 2;`
`grid-column: 2/3;`
 - b. `grid-row: 2;`
`grid-column: 2/4;`
 - c. `row: 2;`
`column: 2/3;`
 - d. `grid-row: 2;`
`column-span: 2/2;`
6. To define a grid layout with areas, use the property:
 - a. `grid-areas`
 - b. `grid-area`
 - c. `grid-template-areas`
 - d. `grid-areas-template`
7. To place an element in the grid area named "intro", apply the style:
 - a. `grid-area: intro;`
 - b. `grid-template-areas: "intro";`
 - c. `area: intro;`
 - d. All of the above
8. To set the space between grid columns to 15 pixels and the space between grid rows to 10 pixels, apply the style:
 - a. `gap: 10px 15px;`
 - b. `grid-gap: 10px 15px;`
 - c. `grid-gap: 15px/10px;`
 - d. `gap: 15px/10px;`

10 15
rows columns
9. To horizontally center the content of a grid cell, apply the style:
 - a. `align-content: center;`
 - b. `align-self: center;`
 - c. `justify-self: center;`
 - d. `justify-content: center;`

Session 3.3 Visual Overview:



Layout with Positioning Styles



Positioning Objects

In the last session, you developed a layout in which page objects were strictly aligned according to the rows and columns of a grid. While a grid layout gives a page a feeling of uniformity and structure, it does limit your freedom to place objects at different locations within the page. In this session, you'll explore how to "break out" of the grid using the CSS positioning styles.

The CSS Positioning Styles

CSS supports several properties to place objects at specific coordinates within the page or within their container. To place an element at a specific position within its container, you use the following style properties

```
position: type;  
top: value;  
right: value;  
bottom: value;  
left: value;
```

where *type* indicates the kind of positioning applied to the element, and the *top*, *right*, *bottom*, and *left* properties indicate the coordinates of the top, right, bottom, and left edges of the element, respectively. The coordinates can be expressed in any of the CSS measuring units or as a percentage of the container's width or height.

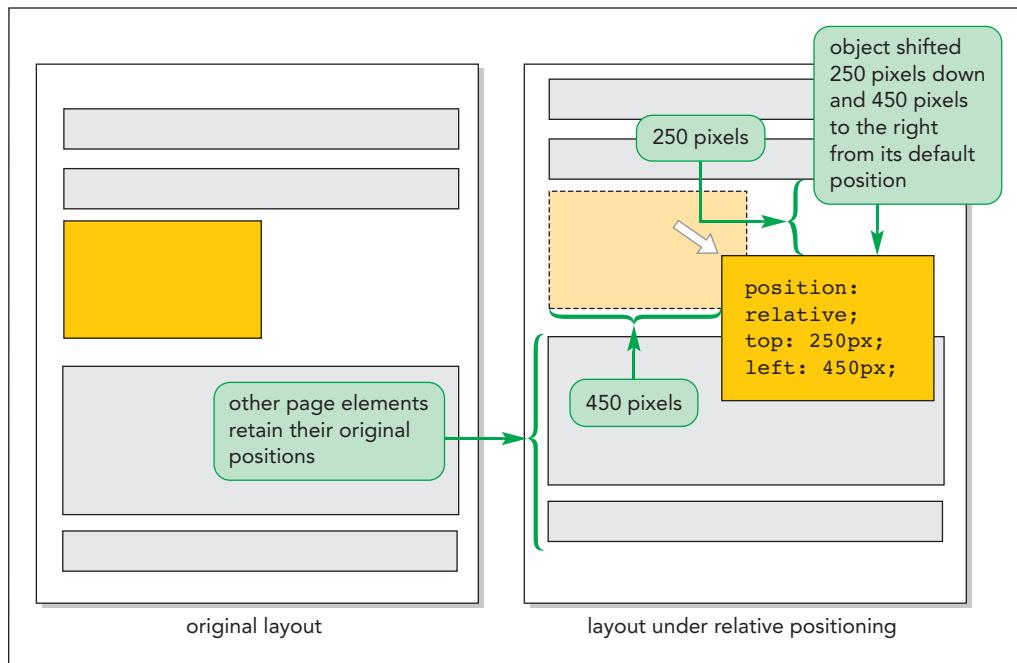
CSS supports five kinds of positioning: *static* (the default), *relative*, *absolute*, *fixed*, and *inherit*. In **static positioning**, the element is placed where it would have fallen naturally within the flow of the document. This is essentially the same as not using any CSS positioning at all. Browsers ignore any values specified for the *top*, *left*, *bottom*, or *right* properties under static positioning.

Relative Positioning

Relative positioning is used to nudge an element out of its normal position in the **document flow**. Under relative positioning, the *top*, *right*, *bottom*, and *left* properties indicate the extra space that is placed alongside the element as it is shifted into a new position. For example, the following style rule adds 250 pixels of space to the top of the element and 450 pixels to the left of the element, resulting in the element being shifted down and to the right (see Figure 3–60):

```
div {  
    position: relative;  
    top: 250px;  
    left: 450px;  
}
```

Figure 3–60 Moving an object using relative positioning



Note that the layout of the other page elements are not affected by relative positioning; they will still occupy their original positions on the rendered page, just as if the object had never been moved at all.

Relative positioning is sometimes used when the designer wants to “tweak” the page layout by slightly moving an object from its default location to a new location that fits the overall page design better. If no top, right, bottom, or left values are specified with relative positioning, their assumed values are 0 and the element will not be shifted at all.

Absolute Positioning

Absolute positioning places an element at specific coordinates within a container where the `top` property indicates the position of the element’s top edge, the `right` property sets the position of the right edge, the `bottom` property sets the bottom edge position, and the `left` property sets the position of the left edge.

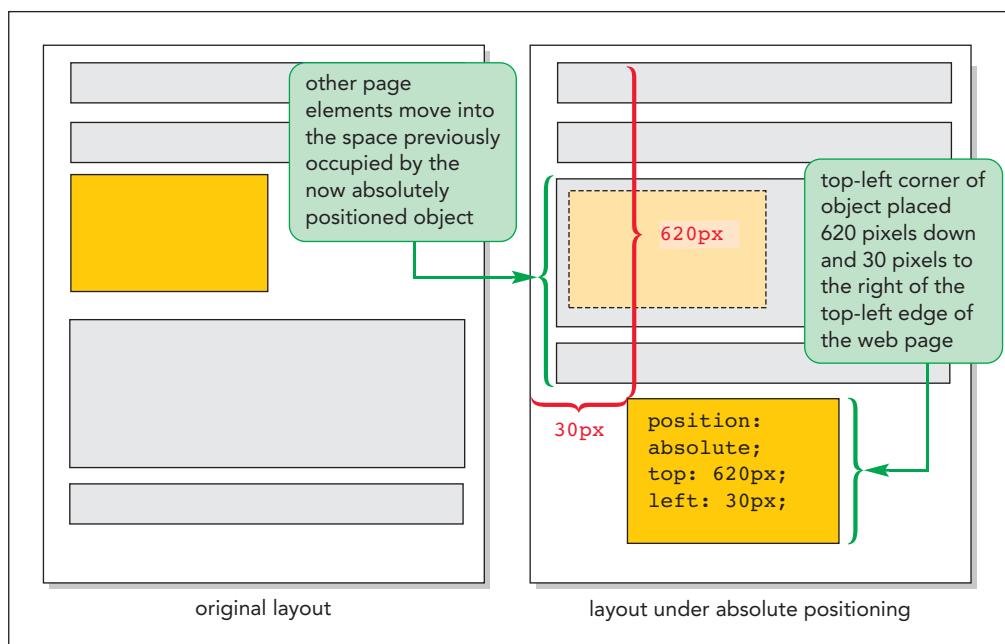
For example, the following style rule places the `header` element 620 pixels from the top edge of its container and 30 pixels from the left edge (see Figure 3–61).

TIP

To place an element at the bottom right corner of its container, use absolute positioning with the right and bottom values set to 0 pixels.

```
header {
    position: absolute;
    top: 620px;
    left: 30px;
}
```

Figure 3–61 Moving an object using absolute positioning



To place an object with absolute positioning, you use either the top/left coordinates or the bottom/right coordinates, but you don't use all four coordinates at the same time because that would confuse the browser. For example an object cannot be positioned along both the left and right edge of its container simultaneously.

As with floating an element, absolute positioning takes an element out of normal document flow with subsequent elements moving into the space previously occupied by the element. This can result in an absolutely positioned object overlapping other page elements.

The interpretation of the coordinates of an absolutely positioned object are all based on the edges of the element's container. Thus the browser needs to "know" where the object's container is before it can absolutely position objects within it. If the container has been placed using a position property set to relative or absolute, the container's location is known and the coordinate values are based on the edges of the container. For example the following style rules place the `article` element at a coordinate that is 50 pixels from the top edge of the `section` element and 20 pixels from the left edge.

```
section {
    position: relative;
}
section > article {
    position: absolute;
    top: 50px;
    left: 20px;
}
```

Note that you don't have to define coordinates for the `section` element as long as you've set its position to relative.

The difficulty starts when the container has not been set using relative or absolute positioning. In that case, the browser has no context for placing an object within the container using absolute positioning. As a result, the browser must go up a level in the hierarchy of page elements, that is, to the container's container. If that container has been placed with absolute or relative positioning, then any object nested within it

TRY IT

You can explore positioning styles using the file `demo_positioning.html` from the `html04 ▶ demo` folder.

can be placed with absolute positioning. For example, in the following style rule, the position of the `article` element is measured from the edges of the `body` element, not the `section` element:

```
body {position: absolute;}

body > section {position: static;}

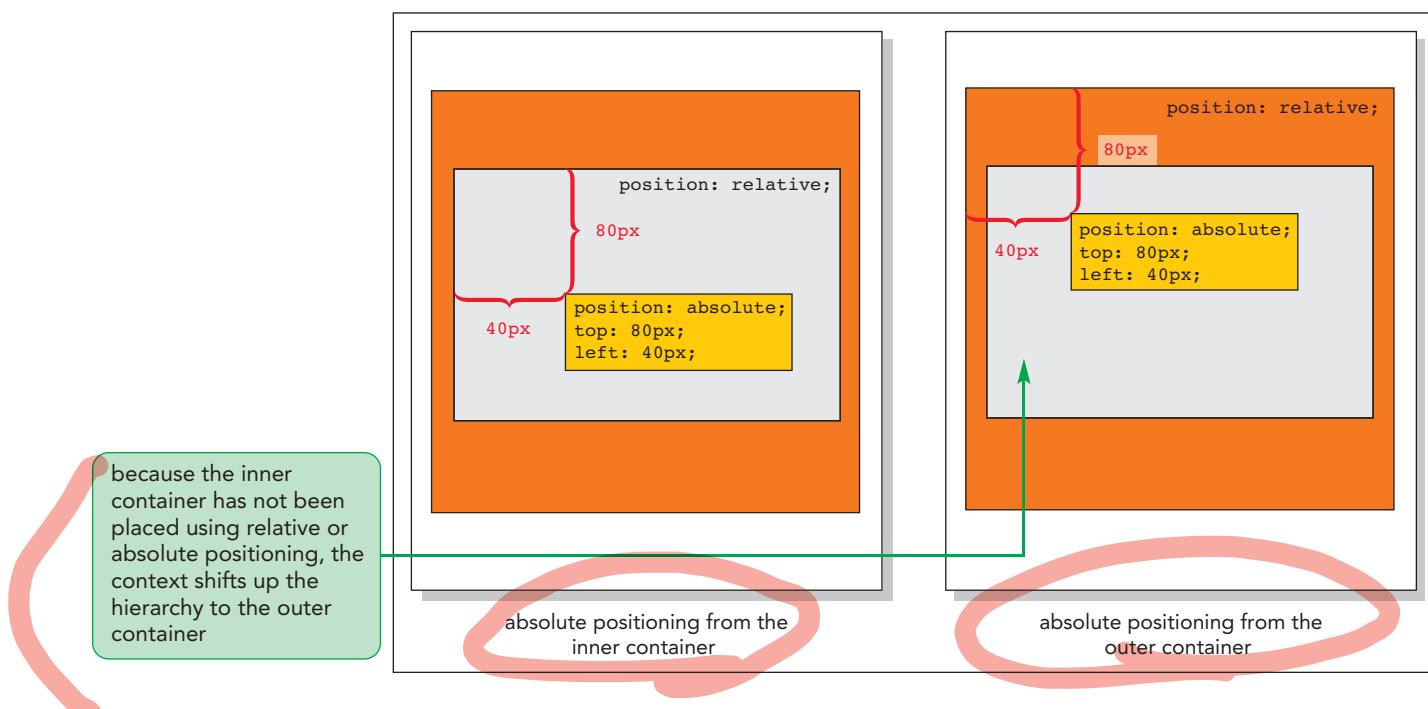
body > section > article {
    position: absolute;
    top: 50px;
    left: 20px;
}
```

TIP

If all of the objects within a container are placed using absolute positioning, the container will have no content and will collapse.

Proceeding in this fashion the browser will continue to go up the hierarchy of elements until it finds a container that has been placed with absolute or relative positioning or it reaches the root `html` element. If it reaches the `html` element, the coordinates of any absolutely positioned object are measured from the edges of the browser window itself. Figure 3–62 shows how the placement of the same object can differ based on which container supplies the context for the `top` and `left` values.

Figure 3–62 Context of the top and left coordinates



Coordinates can be expressed in percentages as well as pixels. Percentages are used for flexible layouts in which the object should be positioned in relation to the width or height of its container. Thus, the following style rule places the `article` element halfway down and 30% to the right of the top-left corner of its container.

```
article {
    position: absolute;
    top: 50%;
    left: 30%;
}
```

As the container of the article changes in width or height, the article's position will automatically change to match.

Fixed and Inherited Positioning

When you scroll through a document in the browser window, the page content scrolls along. If you want to fix an object within the browser window so that it doesn't scroll, you can set its `position` property to `fixed`. For example, the following style rule keeps the footer element at a fixed location, 10 pixels up from the bottom of the browser window:

```
footer {  
    position: fixed;  
    bottom: 10px;  
}
```

Note that a fixed object might cover up other page content, so you should use it with care in your page design.

Finally, you can set the `position` property to `inherit` so that an element inherits the position value of its parent element.

REFERENCE

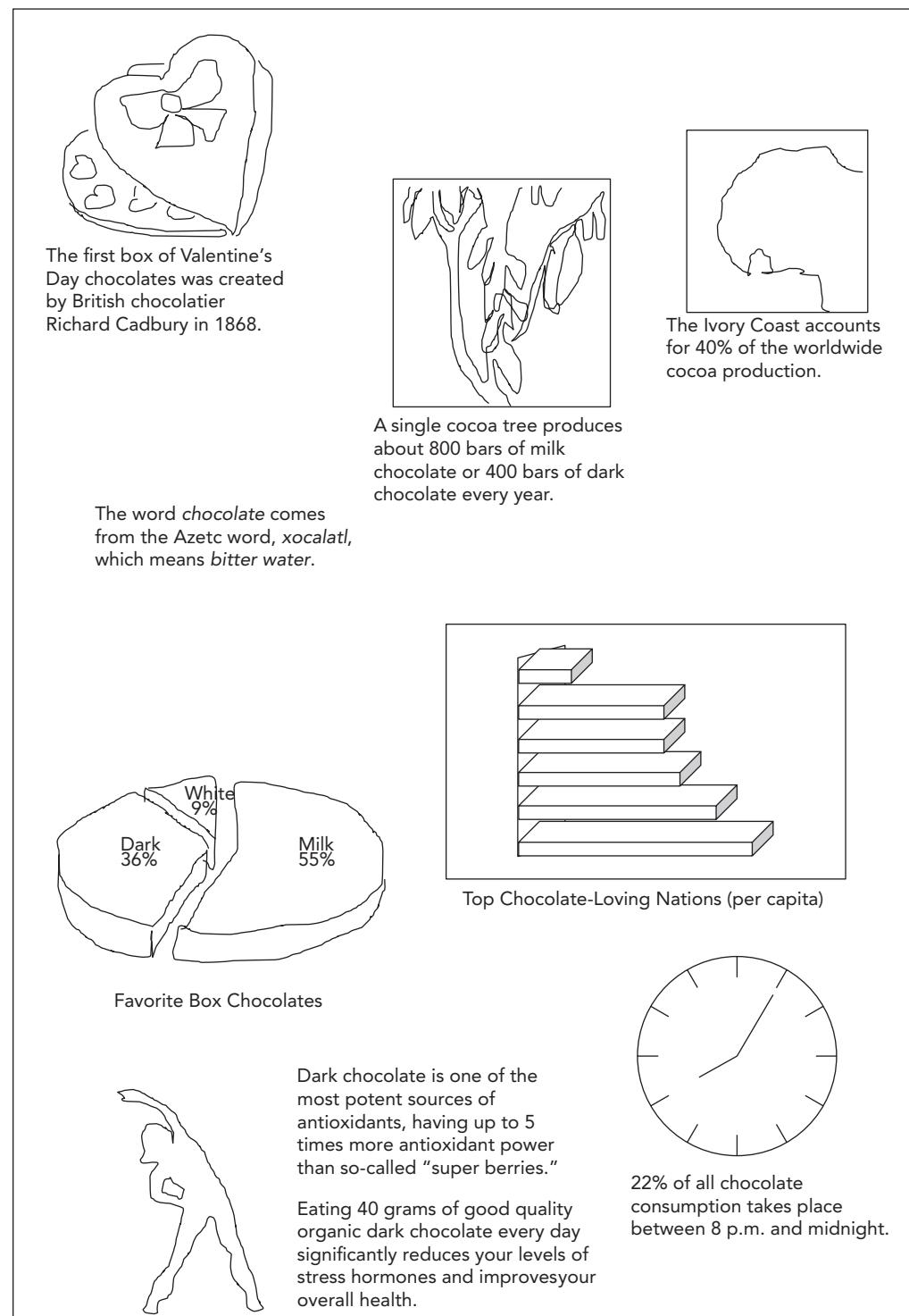
Positioning Objects with CSS

- To shift an object from its default position, use the properties
 - position: relative;
top: value;
left: value;
bottom: value;
right: value;where `value` is the distance in one of the CSS units of measure that the object should be shifted from the corresponding edge of its container.
- To place an object at a specified coordinate within its container, use the properties
 - position: absolute;
top: value;
left: value;
bottom: value;
right: value;where `value` is a distance in one of the CSS units of measure or a percentage of the container's width or height.
- To fix an object within the browser window so that it does not scroll with the rest of the document content, use the property
 - position: fixed;

Using the Positioning Styles

Anne wants you to work on the layout for a page that contains an infographic on chocolate. She sketched the layout of the infographic page, as shown in Figure 3–63.

Figure 3–63 Proposed layout of the chocolate infographic



Because the placement of the text and figures do not line up nicely within a grid, you'll position each graphic and text box using the CSS positioning styles. Anne has already created the content for this page and written the style sheets to format the appearance of the infographic. You will write the style sheet to layout the infographic contents using the CSS positioning styles.

To open the infographic file:

1. Use your editor to open the **pc_info_txt.html** file from the **html03 ▶ tutorial** folder. Enter **your name** and **the date** in the comment section of the file and save the document as **pc_info.html**.
2. Directly after the **title** element, insert the following **link** elements to attach the file to the **pc_reset.css**, **pc_styles3.css**, and **pc_info.css** style sheets.

```
<link href="pc_reset.css" rel="stylesheet" />
<link href="pc_styles3.css" rel="stylesheet" />
<link href="pc_info.css" rel="stylesheet" />
```
3. Take some time to study the structure and content of the **pc_info.html** document. Note that Anne has placed eight information graphics, each within a separate **div** element with a class name of **infobox** and an **id** name ranging from **info1** to **info8**.
4. Close the file, saving your changes.

Next, you'll start working on the **pc_info.css** file, which will contain the positioning and other design styles for the objects in the infographic. You will begin by formatting the **main** element, which contains the infographics. Because you'll want the position of each infographic to be measured from the top-left corner of this container, you will place the **main** element with relative positioning and extend the height of the container to 1400 pixels so that it can contain all eight of the graphic elements.

To format the main element:

1. Use your editor to open the **pc_info_txt.css** file from the **html03 ▶ tutorial** folder. Enter **your name** and **the date** in the comment section of the file and save the document as **pc_info.css**.
2. Go to the Main Styles section and insert the following style rule to format the appearance of the **main** element:

```
main {
    position: relative;
    height: 1400px;
    width: 100%;
}
```

It will be easier to see the effect of placing the different **div** elements if they are not displayed until you are ready to position them. Add a rule to hide the **div** elements, then as you position each element, you can add a style rule to redisplay it.

3. Directly before the Main Styles section, insert the following style rule to hide all of the infoboxes:

```
div.infobox {display:none;}
```

Figure 3–64 highlights the newly added code in the style sheet.

When you want to position objects in an exact or absolute position within a container, set the **position** property of the container to **relative**.

Figure 3–64

Setting the display styles of the main element

```


.infobox {display:none;}


```

```

/* Main Styles */

main {
  position: relative;
  height: 1400px;
  width: 100%;
}

```

- 4. Save your changes to the file and then open the pc_info.html file in your browser. Verify that the browser shows an empty box, about 1400 pixels high, where the infographic will be placed.

Next, you will add a style rule for all of the information boxes so that they are placed within the `main` element using absolute positioning.

To position the information boxes:

- 1. Return to the `pc_info.css` file in your editor and scroll down to the Infographic Styles section.
- 2. Add the following style rule to set the position type of all of the information boxes.

```

div.infobox {
  position: absolute;
}

```

- 3. Within the First Infographic section, add the following style rule to position the first information box 20 pixels from the top edge of its container and 5% from the left edge.

```

div#info1 {
  display: block;
  top: 20px;
  left: 5%;
}

```

Note that we set the `display` property to `block` so that the first information box is no longer hidden on the page. Figure 3–65 highlights the style rules for all of the information boxes and the placement of the first information box.

Figure 3–65

Placing the first information box

```

/* Infographic Styles */

div.infobox {
    position: absolute;
}

/* First Infographic */

div#info1 {
    display: block;
    top: 20px;
    left: 5%;
}

```

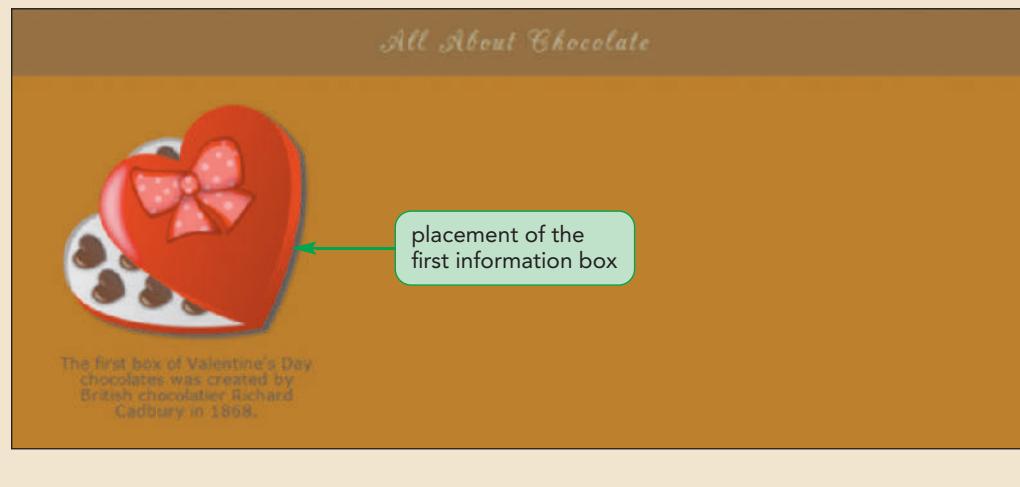
places every information box using absolute positioning

places the first box 20 pixels from the top edge of the main element and 5% from the left

- 4. Save your changes to the file and then reload the `pc_info.html` file in your browser. Figure 3–66 shows the placement of the first information box.

Figure 3–66

Appearance of the first information box



Now place the second and third information boxes.

To place the next two boxes:

- 1. Return to the `pc_info.css` file in your editor and go to the Second Infographic section.
- 2. Add the following style rule to place the second box 185 pixels down from the top of the container and 42% from the left edge.

```

div#info2 {
    display: block;
    top: 185px;
    left: 42%;
}

```

- 3. Within the Third Infographic section insert the following style rule to place the third box 135 pixels from the top edge and 75% of the width of its container from the left edge.

```
div#info3 {
    display: block;
    top: 135px;
    left: 75%;
}
```

Figure 3–67 highlights the style rules to position the second and third information boxes.

Figure 3–67

Positions of the second and third boxes

```
/* Second Infographic */

div#info2 {
    display: block;
    top: 185px;
    left: 42%;
}

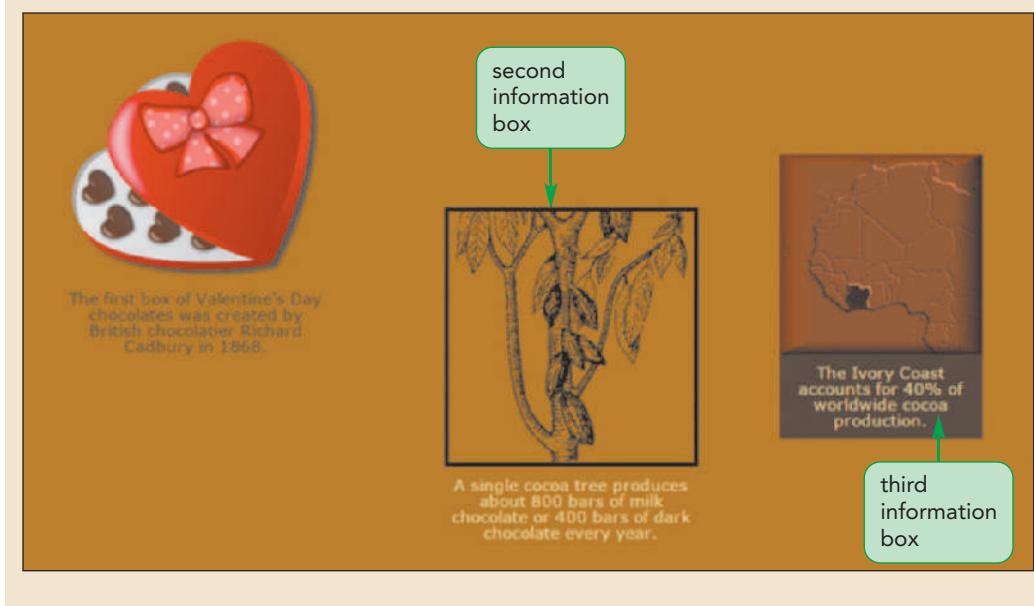
/* Third Infographic */

div#info3 {
    display: block;
    top: 135px;
    left: 75%;
```

- 4. Save your changes to the file and reload pc_info.html in your browser. Figure 3–68 shows the placement of the first three information boxes.

Figure 3–68

Placement of the first three boxes



Place the next three information boxes.

To place the next three boxes:

- 1. Return to the **pc_info.css** file in your editor, go to the Fourth Infographic section and place the fourth box 510 pixels from the top edge and 8% from the left edge.

```
div#info4 {  
    display: block;  
    top: 510px;  
    left: 8%;  
}
```

- 2. Add the following style rule to the Fifth Infographic section to position the fifth box:

```
div#info5 {  
    display: block;  
    top: 800px;  
    left: 3%;  
}
```

- 3. Add the following style rule to the Sixth Infographic section to position the sixth box:

```
div#info6 {  
    display: block;  
    top: 600px;  
    left: 48%;  
}
```

Figure 3–69 highlights the positioning styles for the fourth, fifth, and sixth information boxes.

Figure 3–69

Positions of the fourth, fifth, and sixth boxes

places the fourth box 510 pixels from the top and 8% from the left

places the fifth box 800 pixels from the top and 3% from the left

places the sixth box 600 pixels from the top and 48% from the left

/* Fourth Infographic */

```
div#info4 {
    display: block;
    top: 510px;
    left: 8%;
}
```

/* Fifth Infographic */

```
div#info5 {
    display: block;
    top: 800px;
    left: 3%;
}
```

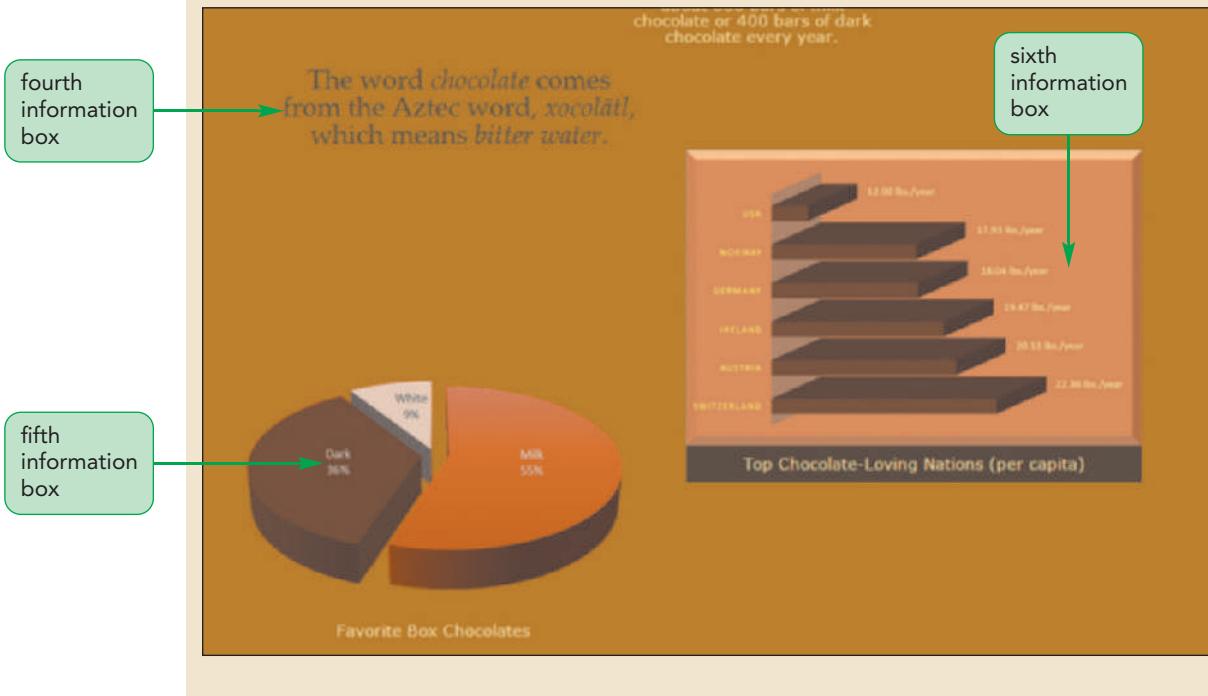
/* Sixth Infographic */

```
div#info6 {
    display: block;
    top: 600px;
    left: 48%;
}
```

- 4. Save your changes to the file and reload pc_info.html in your browser. Figure 3–70 shows the revised layout of the infographic.

Figure 3–70

Placement of the next three boxes



Complete the layout of the infographic by placing the final two boxes on the page.

To place the last two boxes:

- 1. Return to the **pc_info.css** file in your editor, go to the Seventh Infographic section and insert the following style rules:

```
div#info7 {
    display: block;
    top: 1000px;
    left: 68%;
}
```

- 2. Add the following style rules to the Eighth Infographic section:

```
div#info8 {
    display: block;
    top: 1100px;
    left: 12%;
}
```

Figure 3–71 highlights the style rules for the seventh and eighth information boxes.

Figure 3–71

Positioning the seventh and eighth boxes

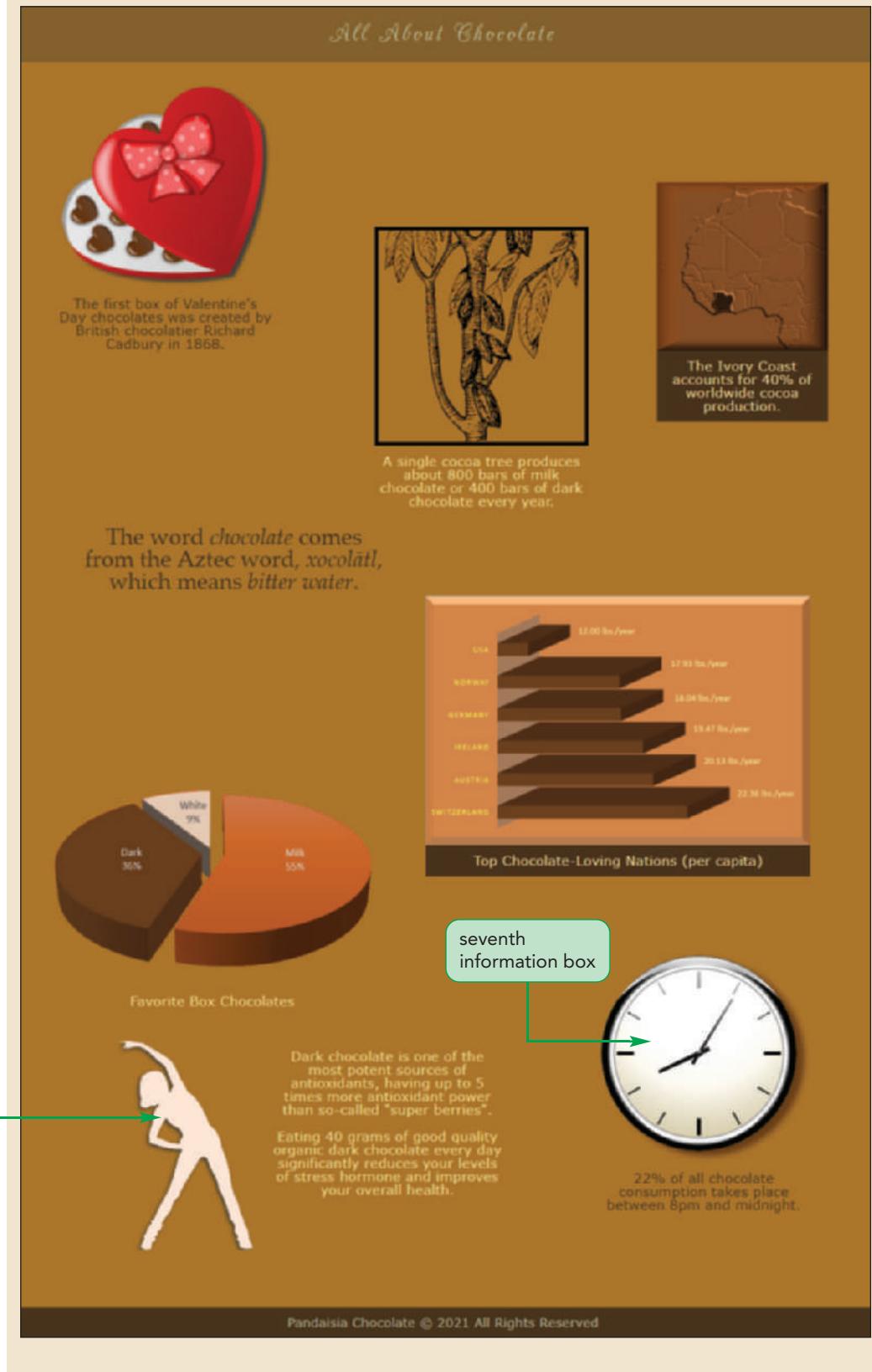
```
/* Seventh Infographic */
div#info7 {
    display: block;
    top: 1000px;
    left: 68%;
}

/* Eighth Infographic */
div#info8 {
    display: block;
    top: 1100px;
    left: 12%;
}
```

- 3. Scroll up to before the Main Styles section and delete the style rule `div.infobox {display: none;}` because you no longer need to hide any information boxes.
- 4. Save your changes to the file and reload `pc_info.html` in your browser. Figure 3–72 show the complete layout of the eight boxes in the infographic.

Figure 3–72

Final layout of the infographic



Anne likes the appearance of the infographic, but she is concerned about its length. She would like you to reduce the height of the infographic so that it appears within the boundaries of the browser window. This change will create overflow because the content is longer than the new height. You will read more about overflow and how to handle it now.

INSIGHT

Creating an Irregular Line Wrap

Many desktop publishing and word-processing programs allow designers to create irregular line wraps in which the text appears to flow tightly around an image. This is not easily done in a web page layout because all images appear as rectangles rather than as irregularly shaped objects. However, with the aid of a graphics package, you can simulate an irregularly shaped image.

The trick is to use your graphics package to slice the image horizontally into several pieces and then crop the individual slices to match the edge of the image you want to display. Once you've edited all of the slices, you can use CSS to stack the separate slices by floating them on the left or right margin, displaying each slice only after the previous slice has been cleared. For example, the following style rule stacks all inline images that belong to the "slice" class on the right margin:

```
img.slice {  
    clear: right;  
    float: right;  
    margin-top: 0px;  
    margin-bottom: 0px;  
}
```

Now any text surrounding the stack of images will tightly match the image's boundary, creating the illusion of an irregular line wrap. Note that you should always set the top and bottom margins to 0 pixels so that the slices join together seamlessly.

Handling Overflow

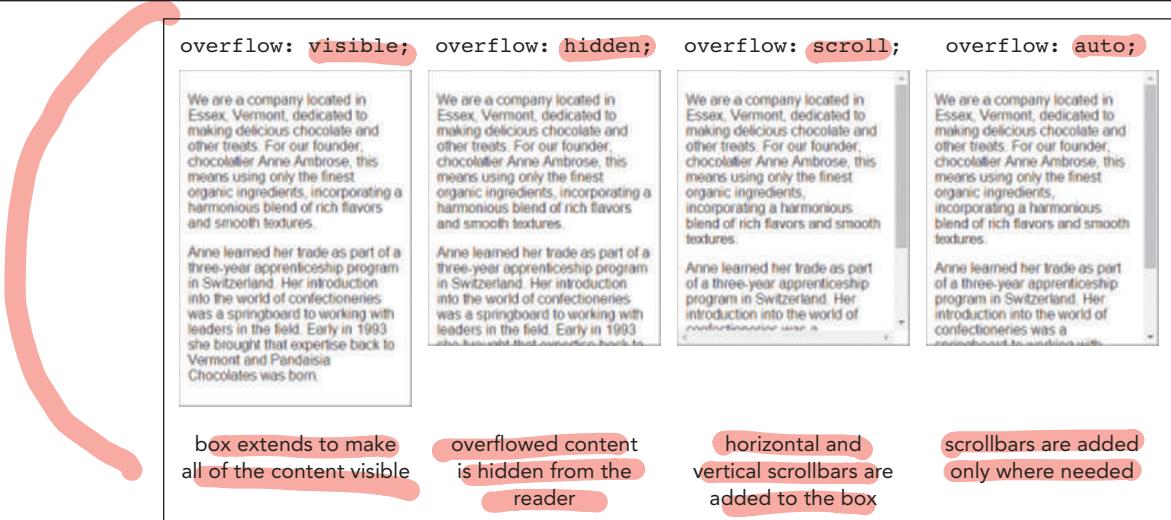
The infographic is long because it displays several information boxes. If you reduce the height of the infographic you run the risk of cutting off several of the boxes that will no longer fit within the reduced infographic. However you can control how your browser handles this excess content using the following `overflow` property

`overflow: type;`

where `type` is `visible` (the default), `hidden`, `scroll`, or `auto`. A value of `visible` instructs browsers to increase the height of an element to fit the overflow content. The `hidden` value keeps the element at the specified height and width, but cuts off excess content. The `scroll` value keeps the element at the specified dimensions, but adds horizontal and vertical scroll bars to allow users to scroll through the overflowed content. Finally, the `auto` value keeps the element at the specified size, adding scroll bars only as they are needed. Figure 3-73 shows examples of the effects of each overflow value on content that is too large for its space.

Figure 3–73

Values of the overflow property



CSS also provides the `overflow-x` and `overflow-y` properties to handle overflow specifically in the horizontal and vertical directions.

REFERENCE

Working with Overflow

- To specify how the browser should handle content that overflows the element's boundaries, use the property
`overflow: type;`
 where `type` is `visible` (the default), `hidden`, `scroll`, or `auto`.

You decide to limit the height of the infographic to 450 pixels and to set the `overflow` property to `auto` so that browsers displays scroll bars as needed for the excess content.

To apply the `overflow` property:

- 1. Return to the `pc_info.css` file in your editor and go to the Main Styles section.
- 2. Within the style rule for the `main` selector, insert the property `overflow: auto;`.
- 3. Reduce the height of the element from `1400px` to **450px**.

Figure 3–74 highlights the revised code in the style rule.

Figure 3–74

Setting the overflow property

```
/* Main Styles */

main {
    overflow: auto;
    position: relative;
    height: 450px;
    width: 100%;
}
```

displays scrollbars if the content overflows the allotted height

sets the height of the infographic to 450 pixels

- 4. Close the file, saving your changes.
- 5. Reload the pc_info.html file in your browser. As shown in Figure 3–75, the height of the infographic has been reduced to 450 pixels and scrollbars have been added that you can use to view the entire infographic.

Figure 3–75

Final layout of the infographic page



- 6. Close any open files now.

Managing White Space with CSS

INSIGHT
in a horizontal

Scroll bars for overflow content are usually placed vertically so that you scroll down to view the extra content. In some page layouts, however, you may want to view content in a horizontal rather than a vertical direction. You can accomplish this by adding the following style properties to the element:

```
overflow: auto;
white-space: nowrap;
```

The white-space property defines how browsers should handle white space in the rendered document. The default is to collapse consecutive occurrences of white space into a single blank space and to automatically wrap text to a new line if it extends beyond the width of the container. However, you can set the white-space property of the element to nowrap to keep inline content on a single line, preventing line wrapping. With the content thus confined to a single line, browsers will display only horizontal scroll bars for the overflow content. Other values of the white-space property include normal (for default handling of white space), pre (to preserve all white space from the HTML file), and pre-wrap (to preserve white space but to wrap excess content to a new line).

Clipping an Element

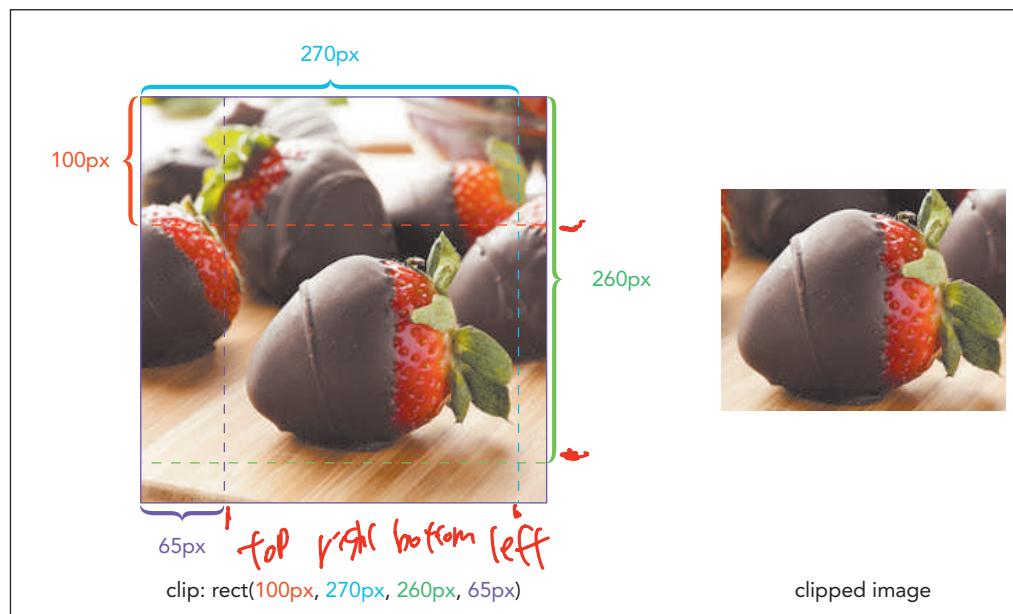
Closely related to the overflow property is the clip property, which defines a rectangular region through which an element's content can be viewed. Anything that lies outside the boundary of the rectangle is hidden. The syntax of the clip property is

```
clip: rect( top, right, bottom, left );
```

where *top*, *right*, *bottom*, and *left* define the coordinates of the clipping rectangle. For example, a clip value of rect(100px, 270px, 260px, 65px) defines a clip region whose top and bottom boundaries are 100 and 260 pixels from the top edge of the element, and whose right and left boundaries are 270 and 65 pixels from the element's left edge. See Figure 3–76.

Figure 3–76

Clipping an image



© Brent Hofacker/Shutterstock.com

The top, right, bottom, and left values also can be set to `auto`, which matches the specified edge of the clipping region to the edge of the parent element. A `clip` value of `rect(10, auto, 125, 75)` creates a clipping rectangle whose right edge matches the right edge of the parent element. To remove clipping completely, apply the style `clip: auto`. Clipping can only be applied when the object is placed using absolute positioning.

REFERENCE

Clipping Content

- To clip an element's content, use the property
`clip: rect(top, right, bottom, left);`
where *top*, *right*, *bottom*, and *left* define the coordinates of the clipping rectangle.
- To remove clipping for a clipped object, use
`clip: auto;`

Stacking Elements

Positioning elements can sometimes lead to objects that overlap each other. By default, elements that are loaded later by the browser are displayed on top of elements that are loaded earlier. In addition, elements placed using CSS positioning are stacked on top of elements that are not. To specify a different stacking order, use the following `z-index` property:

`z-index: value;`

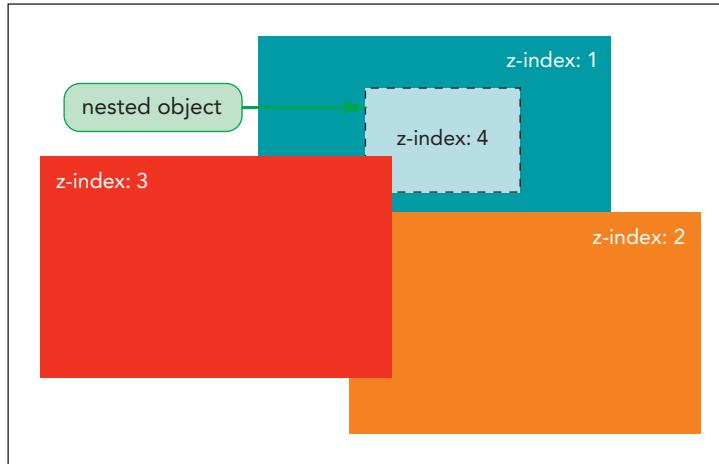
where *value* is a positive or negative integer, or the keyword `auto`. As shown in Figure 3-77, objects with the highest `z-index` values are placed on top of other page objects. A value of `auto` stacks the objects using the default rules.

Figure 3-77 Using the `z-index` property to stack elements



The `z-index` property works only for elements that are placed with absolute positioning. Also, an element's `z-index` value determines its position relative only to other elements that share a common parent; the style has no impact when applied to elements with different parents. Figure 3-78 shows a layout in which the object with a high `z-index` value of 4 is still covered because it is nested within another object that has a low `z-index` value of 1.

Figure 3–78 Stacking nested objects



You do not need to include the `z-index` property in your style sheet because none of the elements in the infographic page are stacked upon another.



Problem Solving: Principles of Design

Good web page design is based on the same common principles found in other areas of art, which include balance, unity, contrast, rhythm, and emphasis. A pleasing layout involves the application of most, if not all, of these principles, which are detailed below:

- **Balance** involves the distribution of elements. It's common to think of balance in terms of **symmetrical balance**, in which similar objects offset each other like items on a balance scale; but you often can achieve more interesting layouts through asymmetrical balance, in which one large page object is balanced against two or more smaller objects.
- **Unity** is the ability to combine different design elements into a cohesive whole. This is accomplished by having different elements share common colors, font styles, and sizes. One way to achieve unity in a layout is to place different objects close to each other, forcing your viewers' eyes to see these items as belonging to a single unified object.
- **Contrast** consists of the differences among all of the page elements. To create an effective design, you need to vary the placement, size, color, and general appearance of the objects in the page so that your viewers' eyes aren't bored by the constant repetition of a single theme.
- **Rhythm** is the repetition or alteration of a design element in order to provide a sense of movement, flow, and progress. You can create rhythm by tiling the same image horizontally or vertically across the page, by repeating a series of elements that progressively increase or decrease in size or spacing, or by using elements with background colors of the same hue but that gradually vary in saturation or lightness.
- **Emphasis** involves working with the focal point of a design. Your readers need a few key areas to focus on. It's a common design mistake to assign equal emphasis to all page elements. Without a focal point, there is nothing for your viewers' eyes to latch onto. You can give a page element emphasis by increasing its size, by giving it a contrasting color, or by assigning it a prominent position in the page.

Designers usually have an intuitive sense of what works and what doesn't in page design, though often they can't say why. These design principles are important because they provide a context in which to discuss and compare designs. If your page design doesn't feel like it's working, evaluate it in light of these principles to identify where it might be lacking.

Anne is pleased with the final design of the infographic page and all of the other pages you've worked on. She'll continue to develop the website and test her page layouts under different browsers and screen resolutions. She'll get back to you with future projects as she continues the redesign of the Pandaisia Chocolates website.

REVIEW**Session 3.3 Quick Check**

1. To shift an object from its default placement in the document flow but keep it within the document flow, use:

- a. absolute positioning
- b.** relative positioning
- c. fixed positioning
- d. static positioning

2. Provide a style to shift rule to shift an article element 15 pixels to the left of its default position in the document flow.

- a.

```
article {  
    position: absolute;  
    left: 15px;  
}
```
- b.**

```
article {  
    position: relative; ✓  
    left: 15px;  
}
```
- c.

```
article {  
    position: absolute;  
    left: -15px;  
}
```
- d.

```
article {  
    position: relative; ✓  
    left: -15px;  
}
```

3. Provide a style to place an article element 15 pixels up from the top edge of its container element.

- a.**

```
article {  
    position: absolute;  
    top: 15px;  
}
```
- b.

```
article {  
    position: relative;  
    top: 15px;  
}
```
- c.

```
article {  
    position: absolute; ✓  
    top: -15px;  
}
```
- d.

```
article {  
    position: relative;  
    top: -15px;  
}
```

4. To place an object using absolute positioning within its container, the container:

- a. must also have absolute positioning
- b.** must have absolute or relative positioning
- c.** must have static positioning
- d. must not have any position property value

5. Provide a style property to display scrollbars when the element content exceeds the element's boundaries.

- a. overflow: auto;
- b** overflow: scroll;
- c. overflow: scrollbar;
- d. overflow: true;

6. An inline image is 400 pixels wide by 300 pixels high. Provide a style rule to clip this image by 10 pixels on each edge.

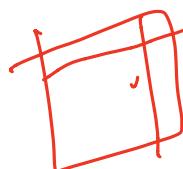
- a** clip: rect(10, 390, 290, 10);
- b. clip: 10;
- c. clip: -10;
- d. clip: 10 390 290 10;

400px

7. If two elements overlap, the one displayed on top will:

- a. be listed first in the document order
- b. have the greater height and width
- c. have the lower z-index value
- d. have the higher z-index value

390 290



more coding / 2-3 coding from chapter that we covered
 20.1 mcq 6
 HTML

Designing for the Mobile Web

Creating a Mobile Website for Daycare Center

not so difficult
 but there might
 be some challenges

OBJECTIVES

Session 5.1

- Create a media query
- Work with the browser viewport
- Apply a responsive design
- Create a pulldown menu with CSS

Session 5.2

- Create a flexbox
- Work with flex sizes
- Explore flexbox layouts

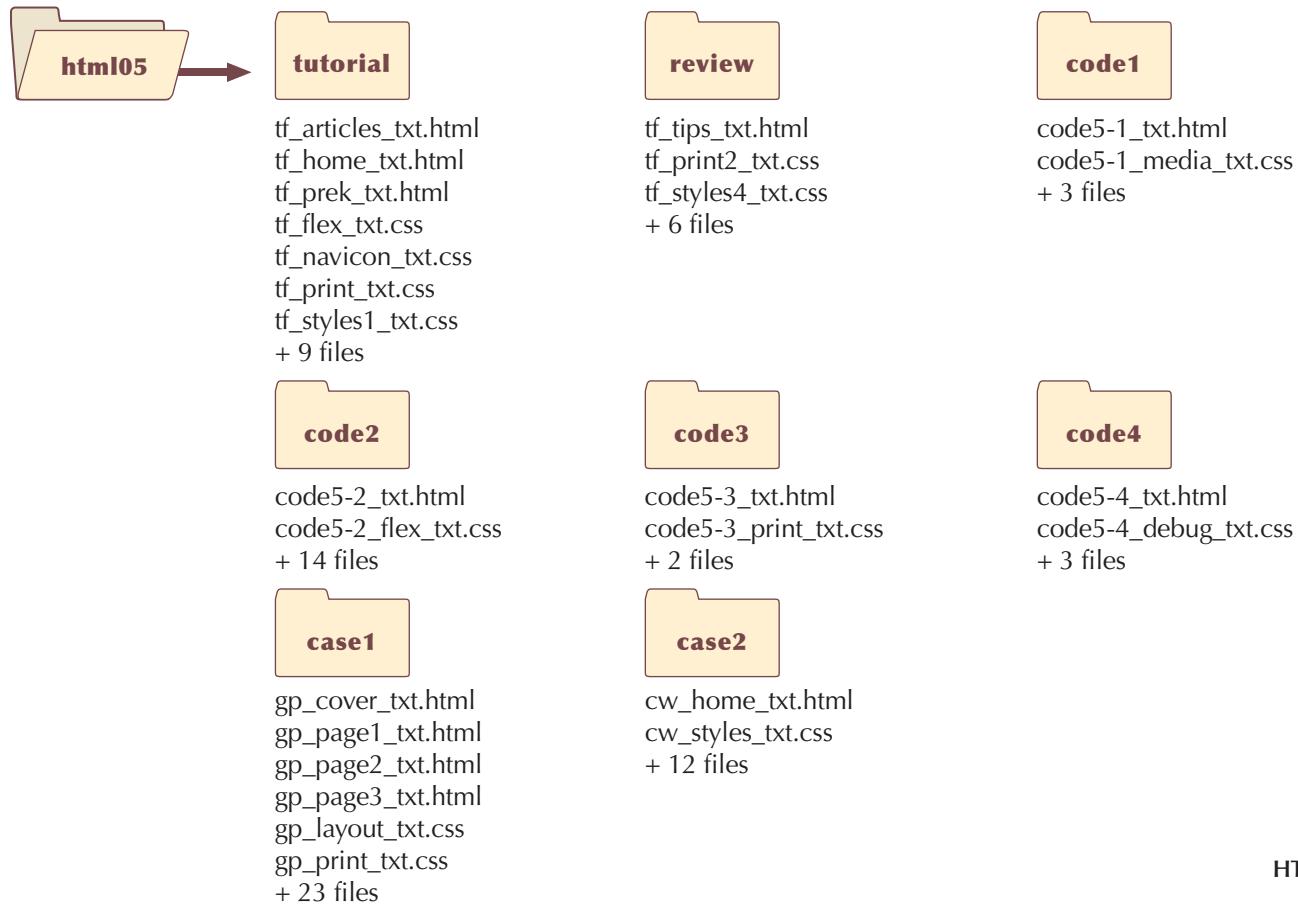
Session 5.3

- Create a print style sheet
- Work with page sizes
- Add and remove page breaks

Case | *Trusted Friends Daycare*

Marjorie Kostas is the owner of *Trusted Friends Daycare*, an early childhood education and care center located in Carmel, Indiana. You've been hired to help work on the redesign of the company's website. Because many of her clients access the website from their mobile phones, Marjorie is interested in improving the site's appearance on mobile devices. However, your design still has to be compatible with tablet devices and desktop computers. Finally, the site contains several pages that her clients will want to print, so your design needs to meet the needs of printed media.

STARTING DATA FILES



Session 5.1 Visual Overview:

The `viewport` meta tag is used to set the properties of the layout viewport.

```
<meta name="viewport" content="width=device-width, initial-scale=1" />
```

This sets the width of the layout viewport equal to the width of the visual viewport.

This sets the initial scale of the viewport to 1.0.

Responsive designs should start with base styles that apply to all devices, followed by mobile styles, tablet styles, and then desktop styles.

A `media query` is used to apply specified style rules to a device based on the device type and the device features.

```
/*
=====
Base Styles
=====
*/
style rules

/*
=====
Mobile Styles: 0 to 480 pixels
=====
*/
@media only screen and (max-width: 480px) {
    style rules
}

/*
=====
Tablet Styles: 481px and greater
=====
*/
@media only screen and (min-width: 481px) {
    style rules
}

/*
=====
Desktop Styles: 769px and greater
=====
*/
@media only screen and (min-width: 769px) {
    style rules
}
```

Within a media query are style rules that are only applied to devices that match the query.

This media query matches screens with a maximum width of 480 pixels.

This media query matches screens with a minimum width of 481 pixels.

This media query matches screens with a minimum width of 769 pixels.

Media Queries

Mobile styles
Tablet styles
Desktop styles



dotshock/Shutterstock.com; Robert Kneschke/Shutterstock.com; Jmlevick/openclipart; Easy/openclipart; BenBois/openclipart

Introducing Responsive Design

In the first four tutorials, you created a single set of layout and design styles for your websites without considering what type of device would be rendering the site. However, this is not always a practical approach and with many users increasingly accessing the web through mobile devices, a web designer must take into consideration the needs of those devices. Figure 5–1 presents some of the important ways in which designing for the mobile experience differs from designing for the desktop experience.

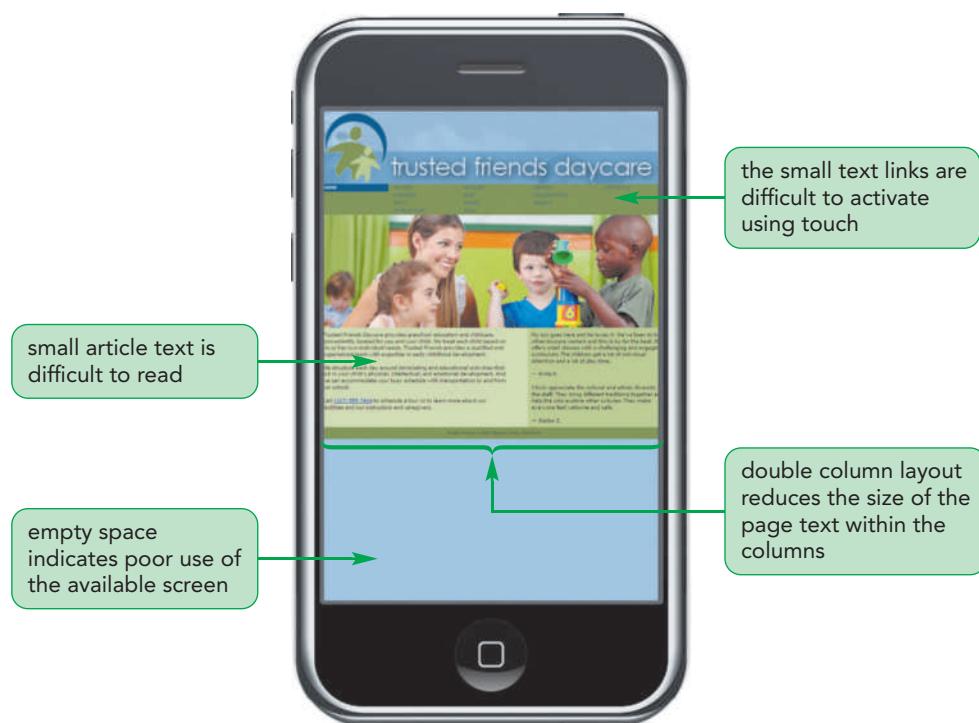
Figure 5–1 Designing for mobile and desktop devices

User Experience	Mobile	Desktop
Page Content	Content should be short and to the point.	Content can be extensive, giving readers the opportunity to explore all facets of the topic.
Page Layout	Content should be laid out within a single column with no horizontal scrolling.	With a wider screen size, content can be more easily laid out in multiple columns.
Hypertext Links	Links need to be easily accessed via a touch interface.	Links can be activated more precisely using a cursor or mouse pointer.
Network Bandwidth	Sites tend to take longer to load over cellular networks and thus overall file size should be kept small.	Sites are quickly accessed over high-speed networks, which can more easily handle large file sizes.
Lighting	Pages need to be easily visible in outdoor lighting through the use of contrasting colors.	Pages are typically viewed in an office setting, allowing a broader color palette.
Device Tools	Mobile sites often need access to devices such as phone dialing, messaging, mapping, and built-in cameras and video.	Sites rarely have need to access desktop devices.

Viewing a web page on a mobile device is a fundamentally different experience than viewing the same web page on a desktop computer. As a result, these differences need to be taken into account when designing a website. Figure 5–2 shows the current home page of the Trusted Friends website as it appears on a mobile device.

Figure 5–2

Trusted Friends home page displayed on a mobile device



© Robert Kneschke/Shutterstock.com; BenBois/openclipart

Notice that the mobile device has automatically zoomed out to display the complete page width resulting in text that is difficult to read and small hypertext links that are practically unusable with a touch interface. While the design might be fine for a desktop monitor in landscape orientation, it's clear that it is ill-suited to a mobile device.

TIP

For more information on the development of responsive design, refer to *Responsive Web Design* by Ethan Marcotte (<http://alistapart.com/article/responsive-web-design>).

What this website requires is a design that is not only specifically tailored to the needs of her mobile users but also is easily revised for tablet and desktop devices. This can be accomplished with responsive design in which the design of the document changes in response to the device rendering the page. An important leader in the development of responsive design is Ethan Marcotte, who identified three primary components of responsive design theory:

- **flexible layout** so that the page layout automatically adjusts to screens of different widths
- **responsive images** that rescale based on the size of the viewing device
- **media queries** that determine the properties of the device rendering the page so that appropriate designs can be delivered to specific devices

In the preceding tutorials, you've seen how to create grid-based fluid layouts and you've used images that scaled based on the width of the browser window and web page. In this session, you'll learn how to work with media queries in order to create a truly responsive website design.

Introducing Media Queries

Media queries are used to associate a style sheet or style rule with a specific device or list of device features. To create a media query within an HTML file, add the following `media` attribute to either the `link` or `style` element in the document head

```
media="devices"
```

where *devices* is a comma-separated list of supported media types associated with a specified style sheet. For example, the following `link` element accesses the `output.css` style sheet file but only when the device is a printer or projection device:

```
<link href="output.css" media="print, projection" />
```

If any other device accesses this web page, it will not load the `output.css` style sheet file. Figure 5–3 lists other possible media type values for the `media` attribute.

Figure 5–3

Media types



Media Type	Used For
all	All output devices (the default)
braille	Braille tactile feedback devices
embossed	Paged Braille printers
handheld	Mobile devices with small screens and limited bandwidth
print	Printers
projection	Projectors
screen	Computer screens
speech	Speech and sound synthesizers, and aural browsers
tty	Fixed-width devices such as teletype machines and terminals
tv	Television-type devices with low resolution, color, and limited scrollability

When no `media` attribute is used, the style sheet is assumed to apply to all devices accessing the web page.

The @media Rule

Media queries can also be used to associate specific style rules with specific devices by including the following `@media` rule in a CSS style sheet file

```
@media devices {
    style rules
}
```

where *devices* are supported media types and *style rules* are the style rules associated with those devices. For example, the following style sheet is broken into three sections: an initial style rule that sets the font color of all `h1` headings regardless of device, a second section that sets the font size for `h1` headings on screen or television devices, and a third section that sets the font size for `h1` headings that are printed:

```
h1 {
    color: red;
}
@media screen, tv {
    h1 {font-size: 2em;}
}
@media print {
    h1 {font-size: 16pt;}
}
```

Note that in this style sheet, the font size for screen and television devices is expressed using the relative `em` unit but the font size for print devices is expressed using points, which is a more appropriate sizing unit for that medium.

Finally, you can specify media devices when importing one style sheet into another by adding the media type to the `@import` rule. Thus, the following CSS rule imports the `screen.css` file only when a screen or projection device is being used:

```
@import url("screen.css") screen, projection;
```

The initial hope was that media queries could target mobile devices using the `handheld` device type; however, as screen resolutions improved to the point where the cutoff between mobile, tablet, laptop, and desktop was no longer clear, media queries began to be based on what features a device supported and not on what the device was called.

Media Queries and Device Features

To target a device based on its features, you add the feature and its value to the `media` attribute using the syntax:

```
media="devices and | or (feature:value)"
```

where `feature` is the name of a media feature and `value` is the feature's value. The `and` and `or` keywords are used to create media queries that involve different devices or different features, or combinations of both.

The `@media` and `@import` rules employ similar syntax:

```
@media devices and|or (feature:value) {  
    style rules  
}
```

and

```
@import url(url) devices and|or (feature:value);
```

For example, the following media query applies the style rules only for screen devices with a width of 320 pixels.

```
@media screen and (device-width: 320px) {  
    style rules  
}
```

Figure 5–4 provides a list of the device features supported by HTML and CSS.

Figure 5–4

Media features

Feature	Description
<code>aspect-ratio</code>	The ratio of the width of the display area to its height
<code>color</code>	The number of bits per color component of the output device; if the device does not support color, the value is 0
<code>color-index</code>	The number of colors supported by the output device
<code>device-aspect-ratio</code>	The ratio of the <code>device-width</code> value to the <code>device-height</code> value
<code>device-height</code>	The height of the rendering surface of the output device
<code>device-width</code>	The width of the rendering surface of the output device
<code>height</code>	The height of the display area of the output device
<code>monochrome</code>	The number of bits per pixel in the device's monochrome frame buffer
<code>orientation</code>	The general description of the aspect ratio: equal to <code>portrait</code> when the height of the display area is greater than the width; equal to <code>landscape</code> otherwise
<code>resolution</code>	The resolution of the output device in pixels, expressed in either dpi (dots per inch) or dpcm (dots per centimeter)
<code>width</code>	The width of the display area of the output device

All of the media features in Figure 5–4, with the exception of `orientation`, also accept `min-` and `max-` prefixes, where `min-` provides a minimum value for the specified feature, and `max-` provides the feature's maximum value. Thus, the following media query applies style rules only for screen devices whose width is at most 700 pixels:

```
@media screen and (max-width: 700px) {  
    style rules  
}
```

Similarly, the following media query applies style rules only to screens that are at least 400 pixels wide:

```
@media screen and (min-width: 400px) {  
    style rules  
}
```

You can combine multiple media features using logical operators such as `and`, `not`, and `or`. The following query applies the enclosed styles to all media types but only when the width of the output devices is between 320 and 480 pixels (inclusive):

```
@media all and (min-width: 320px) and (max-width: 480px) {  
    style rules  
}
```

Some media features are directed toward devices that do not have a particular property or characteristic. This is done by applying the `not` operator, which negates any features found in the expression. For example, the following query applies only to media devices that are not screen or do not have a maximum width of 480 pixels:

```
@media not screen and (max-width: 480px) {  
    style rules  
}
```

TIP

If you specify a feature without specifying a device, the media query will apply to all devices.

For some features, you do not have to specify a value but merely indicate the existence of the feature. The following query matches any screen device that also supports color:

```
@media screen and (color) {  
    style rules  
}
```

Finally, for older browsers that do not support media queries, CSS provides the `only` keyword to hide style sheets from those browsers. In the following code, older browsers will interpret `only` as an unsupported device name and so will not apply the enclosed style rules, while newer browsers will recognize the keyword and continue to apply the style rules.

```
@media only screen and (color) {  
    style rules  
}
```

All current browsers support media queries, but you will still see the `only` keyword used in many website style sheets.

Creating a Media Query

- To create a media query that matches a device in a `link` or `style` element within an HTML file, use the following `media` attribute

```
media="devices and|or (feature:value)"
```

where `devices` is a comma-separated list of media types, `feature` is the name of a media feature, and `value` is the feature's value

- To create a media query, create the following `@media` rule within a CSS style sheet

```
@media devices and|or (feature:value) {
    style rules
}
```

where `style rules` are the style rules applied for the specified device and feature.

- To import a style sheet based on a media query, apply the following `@import` rule within a CSS style sheet

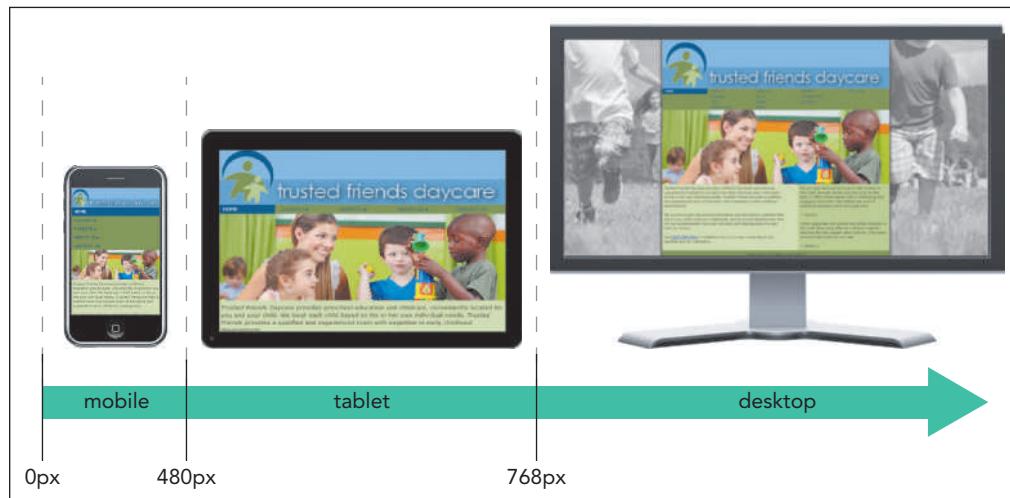
```
@import url(url) devices and|or (feature:value);
```

Applying Media Queries to a Style Sheet

You meet with Marjorie to discuss her plans for the home page redesign. She envisions three designs: one for mobile devices, a different design for tablets, and finally a design for desktop devices based on the current appearance of the site's home page (see Figure 5–5).

Figure 5–5

Trusted Friends home page for different screen widths



© Robert Kneschke/Shutterstock.com; © dotshock/Shutterstock.com; BenBois/openclipart; JMLevick/openclipart; Easy/openclipart

The mobile design will be used for screen widths up to 480 pixels, the tablet design will be used for widths ranging from 481 pixels to 768 pixels, and the desktop design will be used for screen widths exceeding 768 pixels. To apply this approach, you'll create a style sheet having the following structure:

```
/* Base Styles */
style rules

/* Mobile Styles */
@media only screen and (max-width: 480px) {
    style rules
}

/* Tablet Styles */
@media only screen and (min-width: 481px) {
    style rules
}

/* Desktop Styles */
@media only screen and (min-width: 769px) {
    style rules
}
```

Note that this style sheet applies the principle **mobile first** in which the overall page design starts with base styles that apply to all devices followed by style rules specific to mobile devices. Tablet styles are applied when the screen width is 481 pixels or greater, and desktop styles build upon the tablet styles when the screen width exceeds 768 pixels. Thus, as your screen width increases, you add on more features or replace features found in smaller devices. In general, with responsive design, it is easier to add new styles through progressive enhancement than to replace styles.

Marjorie has supplied you with the HTML code and initial styles for her website's home page. Open her HTML file now.

To open the site's home page:

- 1. Use your editor to open the **tf_home_txt.html** and **tf_styles1_txt.css** files from the **html05 ▶ tutorial** folder. Enter **your name** and **the date** in the comment section of each file and save them as **tf_home.html** and **tf_styles1.css** respectively.
- 2. Return to the **tf_home.html** file in your editor and, within the document head, create links to the **tf_reset.css** and **tf_styles1.css** style sheet files.
- 3. Take some time to scroll through the contents of the document to become familiar with its contents and structure and then save your changes to the file, but do not close it.

Next, you'll insert the structure for the responsive design styles in the **tf_styles1.css** style sheet, adding sections for mobile, tablet, and desktop devices.

To add media queries to a style sheet:

- 1. Return to the **tf_styles1.css** file in your editor.
- 2. Marjorie has already inserted the base styles that will apply to all devices at the top of the style sheet file. Take time to review those styles.

3. Scroll to the bottom of the document and add the following code and comments after the New Styles Added Below comment.

```

/*
=====
Mobile Styles: 0px to 480px
=====
*/
@media only screen and (max-width: 480px) {

}

/*
=====
Tablet Styles: 481px and greater
=====
*/
@media only screen and (min-width: 481px) {

}

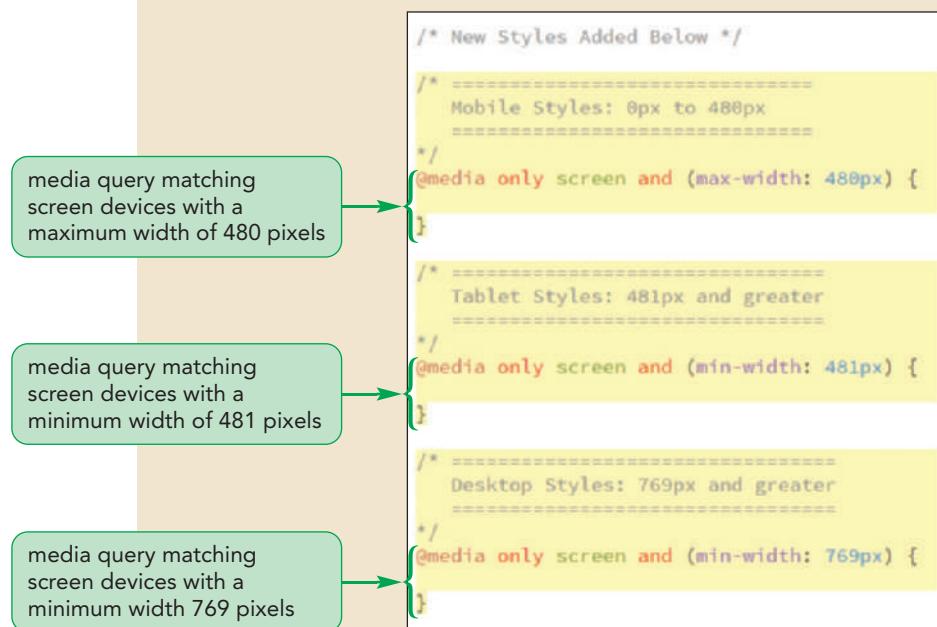
/*
=====
Desktop Styles: 769px and greater
=====
*/
@media only screen and (min-width: 769px) {
}

```

Figure 5–6 highlights the media queries in the style sheet file.

Figure 5–6

Creating media queries for different screen widths



4. Save your changes to the file.

The media queries you've written are based on the screen width. However, before you can begin writing styles for each media query, you have to understand how those width values are interpreted by your browser.

Exploring Viewports and Device Width

Web pages are viewed within a window called the viewport. For desktop computers, the viewport is the same as the browser window; however, this is not the case with mobile devices. Mobile devices have two types of viewports: a **visual viewport** displaying the web page content that fits within a mobile screen and a **layout viewport** containing the entire content of the page, some of which may be hidden from the user.

The two viewports exist in order to accommodate websites that have been written with desktop computers in mind. A mobile device will automatically zoom out of a page in order to give users the complete view of the page's contents, but as shown earlier in Figure 5–2, this often results in a view that is too small to be usable. While the user can manually zoom into a page to make it readable within the visual viewport, this is done at the expense of hiding content, as shown in Figure 5–7.

Figure 5–7 Comparing the **visual** and **layout** viewports



© Robert Kneschke/Shutterstock.com; BenBois/openclipart

Notice in the figure how the home page of the Trusted Friends website has been zoomed in on a mobile device so that only part of the page is displayed within the visual viewport and the rest of the page, which is hidden from the user, extends into the layout viewport.

Widths in media queries are based on the width of the layout viewport, not the visual viewport. Thus, depending on how the page is scaled, a width of 980 pixels might match the physical width of the device as shown in Figure 5–2 or it might extend beyond it as shown in Figure 5–7. In order to correctly base a media query on the

physical width of the device, you have to tell the browser that you want the width of the layout viewport matched to the device width by adding the following `meta` element to the HTML file:

```
<meta name="viewport" content="properties" />
```

where `properties` is a comma-separated list of viewport properties and their values, as seen in the example that follows:

```
<meta name="viewport"
      content="width=device-width, initial-scale=1" />
```

In this `meta` element, the `device-width` keyword is used to set the width of the layout viewport to the physical width of the device's screen. For a mobile device, this command sets the width of the layout viewport to the width of the device. The line `initial-scale=1` is added so that the browser doesn't automatically zoom out of the web page to fit the page content within the width of the screen. We want the viewport to match the device width, which is what the above `meta` element tells the browser to do.

REFERENCE

Configuring the Layout Viewport

- To configure the properties of the layout viewport for use with media queries, add the following `meta` element to the HTML file

```
<meta name="viewport" content="properties" />
```

where `properties` is a comma-separated list of viewport properties and their values.

- To size the layout viewport so that it matches the width of the device without rescaling, use the following `viewport` `meta` element

```
<meta name="viewport"
      content="width=device-width, initial-scale=1" />
```

Add the `viewport` `meta` element to the `tf_home.html` file now, setting the width of the layout viewport to match the device width and the initial scale to 1.

To define the visual viewport:

- 1. Return to the `tf_home.html` file in your editor.
- 2. Below the `meta` element that defines the character set, insert the following HTML tag:

```
<meta name="viewport"
      content="width=device-width, initial-scale=1" />
```

Figure 5-8 highlights the code for the `viewport` `meta` element.

Figure 5–8 Setting the properties of the viewport

page does not automatically zoom out when the page is initially opened by the browser

```

<title>Trusted Friends Daycare</title>
<meta charset="utf-8" />
<meta name="viewport" content="width=device-width, initial-scale=1" />
<link href="tf_reset.css" rel="stylesheet" />
<link href="tf_styles1.css" rel="stylesheet" />

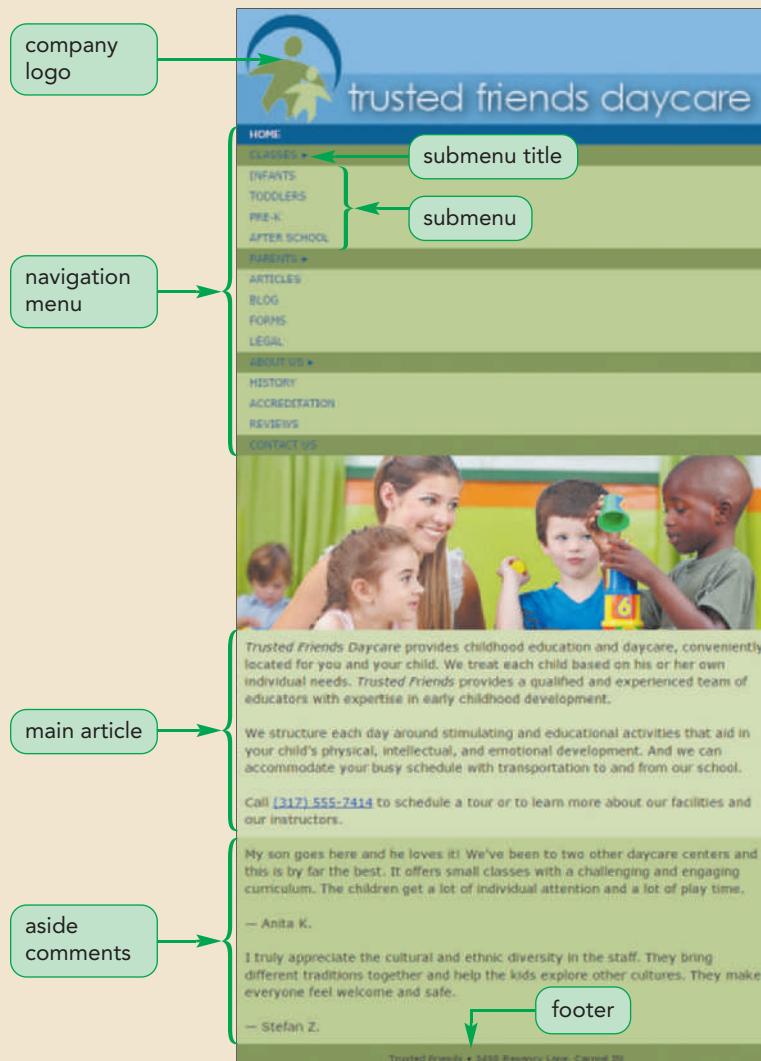
```

sets the width of the layout viewport to the width of the device

- 4. Save your changes to the file.
- 5. Open the **tf_home.html** file in your browser. Figure 5–9 shows the initial design of the page.

Figure 5–9

Mobile layout of the Trusted Friends home page



© Robert Kneschke/Shutterstock.com

Now that you've set up the media queries and configured the viewport, you can work on the design of the home page. You'll start by designing for mobile devices.

INSIGHT

Not All Pixels Are Equal

While pixels are a basic unit of measurement in web design, there are actually two types of pixels to consider as you design a website. One is a **device pixel**, which refers to the actual physical pixel on a screen. The other is a **CSS pixel**, which is the fundamental unit in CSS measurements. The difference between device pixels and CSS pixels is easiest to understand when you zoom into and out of a web page. For example, the following style creates an aside element that is 300 CSS pixels wide:

```
aside: {width: 300px;}
```

However, the element is not necessarily 300 device pixels. If the user zooms into the web page, the apparent size of the article increases as measured by device pixels but remains 300 CSS pixels wide, resulting in 1 CSS pixel being represented by several device pixels.

The number of device pixels matched to a single CSS pixel is known as the **device-pixel ratio**. When a page is zoomed at a factor of 2x, the device-pixel ratio is 2, with a single CSS pixel represented by a 2x2 square of device pixels.

One area where the difference between device pixels and CSS pixels becomes important is in the development of websites optimized for displays with high device-pixel ratios. Some mobile devices are capable of displaying images with a device pixel ratio of 3, resulting in free crisp and clear images. Designers can optimize their websites for these devices by creating one set of style sheets for low-resolution displays and another for high-resolution displays. The high-resolution style sheet would load extremely detailed, high-resolution images, while the low-resolution style sheet would load lower resolution images better suited to devices that are limited to smaller device-pixel ratios. For example, the following media query

```
<link href="retina.css" rel="stylesheet"  
media="only screen and (-webkit-min-device-pixel-ratio: 2) " />
```

loads the *retina.css* style sheet file for high-resolution screen devices that have device-pixel ratios of at least 2. Note that currently the *device-pixel-ratio* feature is a browser-specific extension supported only by WebKit.

Creating a Mobile Design

A mobile website design should reflect how users interact with their mobile devices. Because your users will be working with a small handheld touchscreen device, one key component in your design is to have the most important information up-front and easily accessible, which means your home page on a mobile device needs to be free of unnecessary clutter. Another important principle of designing for mobile devices is that you should limit the choices you offer to your users. Ideally, there should only be a few navigation links on the screen at any one time.

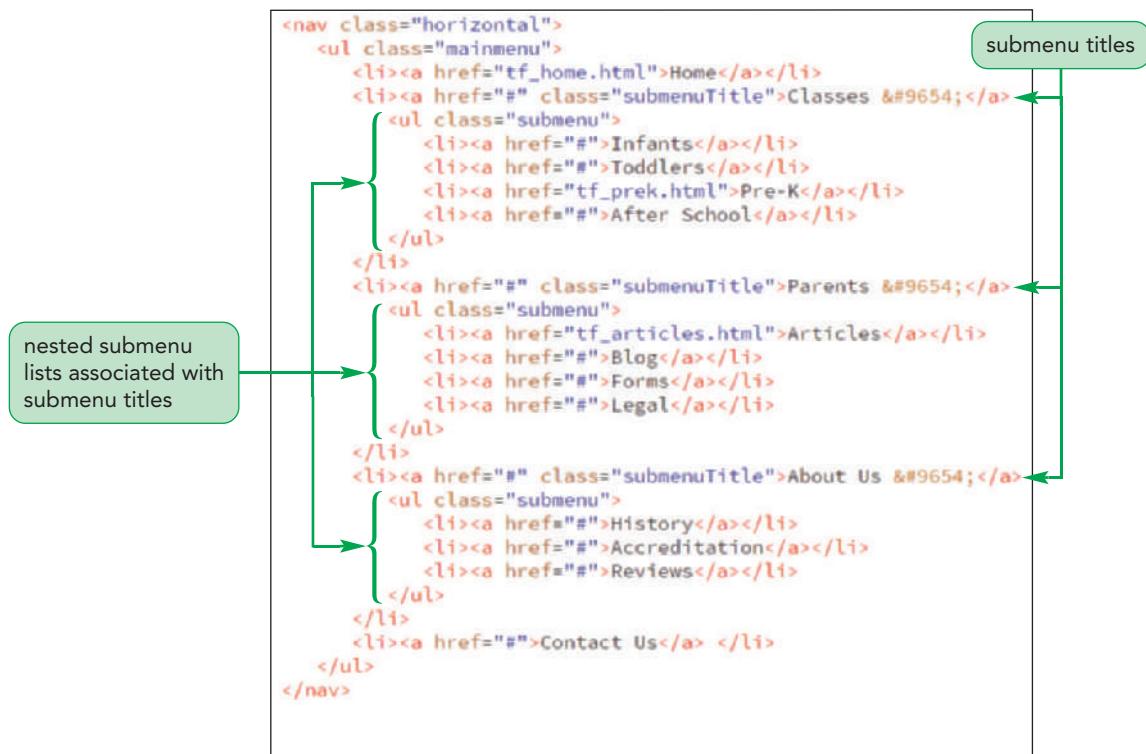
With these principles in mind, consider the current layout of the Trusted Friends home page shown in Figure 5–9. The content is arranged within a single column providing the maximum width for the text and images, but an area of concern for Marjorie is the long list of hypertext links, which forces the user to scroll vertically down the page to view information about the center. Most mobile websites deal with this issue by hiding extensive lists of links in pulldown menus, appearing only in response to a tap of a major heading in the navigation list. You'll use this technique for the Trusted Friends home page.

Creating a Pulldown Menu with CSS

Marjorie has already laid the foundation for creating a pulldown menu in her HTML code. Figure 5–10 shows the code used to mark the contents of the navigation list in the body header.

Figure 5–10

Submenus in the navigation list



Marjorie has created a navigation bar that includes topical areas named Classes, Parents, and About Us. Within each of these topical areas are nested lists containing links to specific pages on the Trusted Friends website. Marjorie has put each of these nested lists within a class named *submenu*. So, first you'll hide each of these submenus to reduce the length of the navigation list as it is rendered within the user's browser. You'll place this style rule in the section for Base Styles because it will be used by both mobile and tablet devices (but not by desktop devices as you'll see later).

To hide a submenu:

- 1. Return to the **tf_styles1.css** file in your editor.
- 2. Scroll to the Pulldown Menu Styles section and add the following style rule:

```

ul.submenu {
  display: none;
}

```

Figure 5–11 highlights the styles to hide the navigation list submenus.

Figure 5–11

Hiding the navigation list submenus

prevents the submenu unordered lists from being displayed

```
/* Pulldown Menu Styles */
ul.submenu {
    display: none;
}
```

- 3. Save your changes to the file and then reload the `tf_home.html` file in your browser. Verify that the navigation list no longer shows the contents of the submenus but only the Home, Classes, Parents, About Us, and Contact Us links. See Figure 5–12.

Figure 5–12

Navigation list with hidden submenus



© Robert Kneschke/Shutterstock.com

Next, you want to display a nested submenu only when the user hovers the mouse pointer over its associated submenu title, which for this page are the Classes, Parents, and About Us titles. Because the submenu follows the submenu title in the HTML file (see Figure 5–10), you can use the following selector to select the submenu that is immediately preceded by a hovered submenu title:

```
a.submenuTitle:hover+ul.submenu
```

However, this selector is not enough because you want the submenu to remain visible as the pointer moves away from the title and hovers over the now-visible submenu. So, you need to add `ul.submenu:hover` to the selector:

```
a.submenuTitle:hover+ul.submenu, ul.submenu:hover
```

To make the submenu visible, you change its display property back to `block`, resulting in the following style rule:

```
a_submenuTitle:hover+ul.submenu, ul.submenu:hover {
    display: block;
}
```

You may wonder why you don't use only the `ul.submenu:hover` selector. The reason is that you can't hover over the submenu until it's visible and it won't be visible until you first hover over the submenu title. Add this rule now to the `tf_styles1.css` style sheet and test it.

To redisplay the navigation submenus:

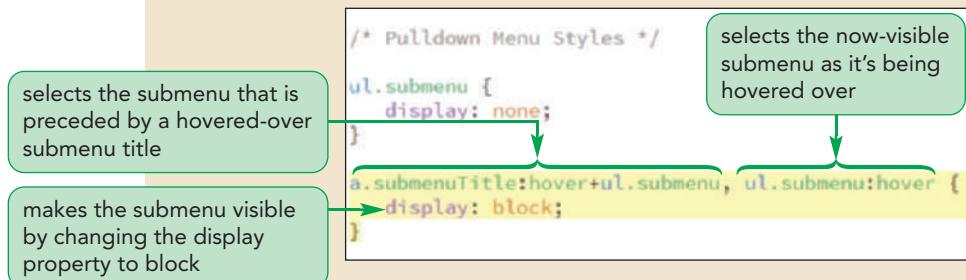
- 1. Return to the `tf_styles1.css` file in your editor.
- 2. Add the following style rule to the Pulldown Menu Styles section:

```
a_submenuTitle:hover+ul.submenu, ul.submenu:hover {
    display: block;
}
```

Figure 5–13 highlights the styles to display the navigation list submenus.

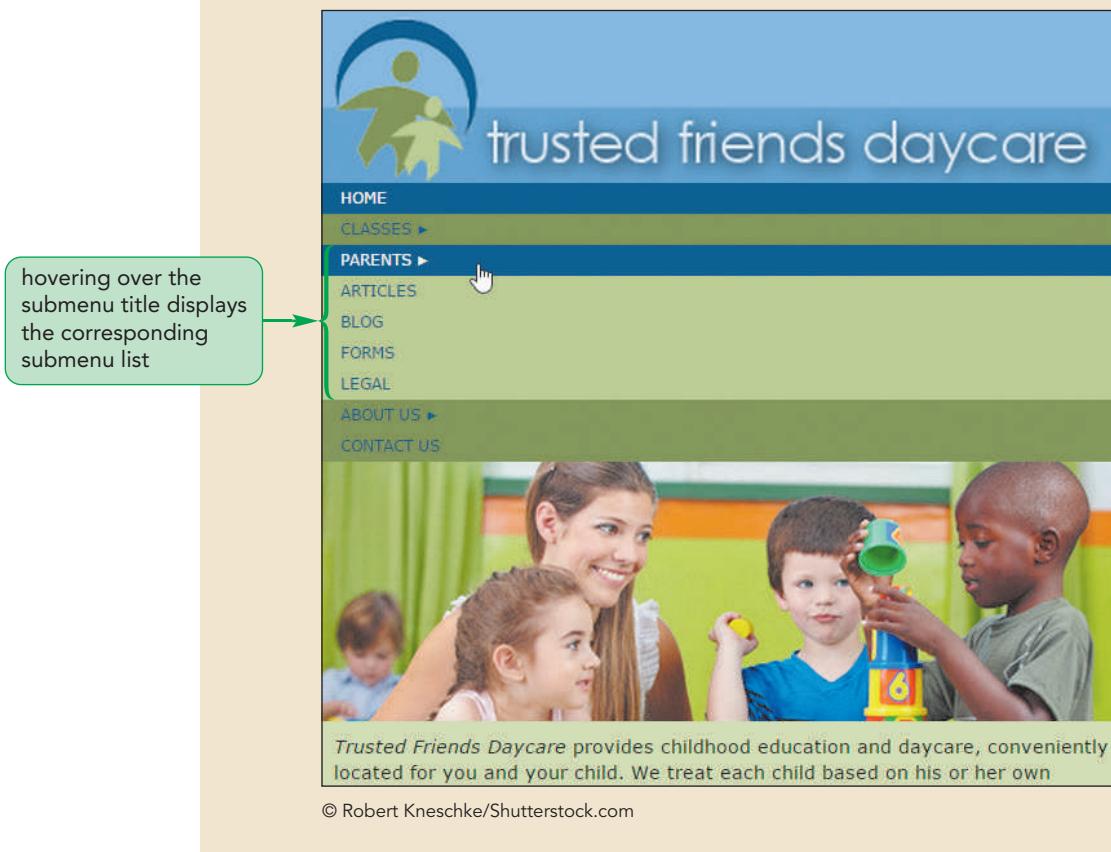
Figure 5–13

Displaying the hidden submenus



- 3. Save your changes to the file and then reload the `tf_home.html` file in your browser. Hover your mouse pointer over each of the submenu titles and verify that the corresponding submenu becomes visible and remains visible as you move the mouse pointer over its contents.

Figure 5–14 shows the revised appearance of the navigation list using the pulldown menus.

Figure 5–14 Displaying the contents of a pulldown menu

The hover event is used with mouse pointers on desktop computers, but it has a different interpretation when applied to mobile devices. Because almost all mobile devices operate via a touch interface, there is no hovering. A mobile browser will interpret a hover event as a tap event in which the user taps the page object. When the hover event is used to hide an object or display it (as we did with the submenus), mobile browsers employ a double-tap event in which the first tap displays the page object and a second tap, immediately after the first, activates any hypertext links associated with the object. To display the Trusted Friends submenus, the user would tap the submenu title and to hide the submenus the user would tap elsewhere on the page.

To test the hover action, you need to view the Trusted Friends page on a mobile device or a mobile emulator.

Testing Your Mobile Website

The best way to test a mobile interface is to view it directly on a mobile device. However, given the large number of mobile devices and device versions, it's usually not practical to do direct testing on all devices. An alternative to having the physical device is to emulate it through a software program or an online testing service. Almost every mobile phone company provides a software development kit or SDK that developers can use to test their programs and websites. Figure 5–15 lists some of the many **mobile device emulators** available on the web at the time of this writing.

Figure 5–15 Popular device emulators

Mobile Emulator	Description
Android SDK	Software development kit for Android developers (developer.android.com/sdk)
iOS SDK	Software development kit for iPhone, iPad, and other iOS devices (developer.apple.com)
Mobile Phone Emulator	Online emulation for a variety of mobile devices (www.mobilephoneemulator.com)
Mobile Test Me	Online emulation for a variety of mobile devices (mobiletest.me)
Opera Mobile SDK	Developer tools for the Opera Mobile browser (www.opera.com/developer)

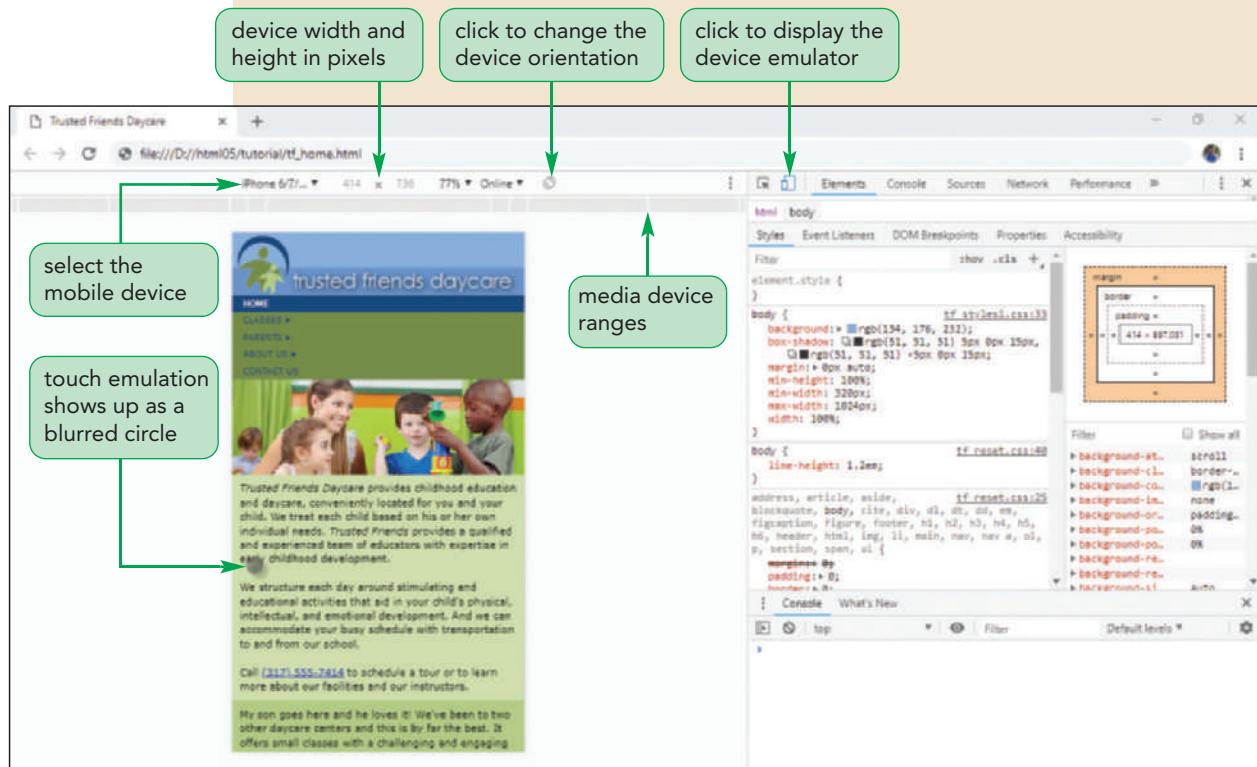
Browsers are also starting to include device emulators as part of their developer tools. You will examine the device emulator that is supplied with the Google Chrome browser and use it to view the Trusted Friends home page under a device of your choosing. If you don't have access to the Google Chrome browser, review the steps that follow and apply them to the emulator of your choice.

Viewing the Google Chrome device emulator:

- 1. Return to the **tf_home.html** file in the Google Chrome browser and press **F12** to open the developer tools pane.
 - 2. If necessary, click the **device** icon  located at the top of the developer pane to display the device toolbar.
 - 3. Select a device of your choosing from the drop-down list of devices on the developer toolbar.
 - 4. Refresh or reload the web page to ensure that the display parameters of your selected device are applied to the rendered page.
- The emulator also allows you to view the effect of changing the orientation of the phone from portrait to landscape.
- 5. Click the **rotate** button  located on the device toolbar to switch to landscape orientation. Click the **rotate** button again to switch back to portrait mode.
- Google Chrome's device emulator can also emulate the touch action. The touch point is represented by a semitransparent circle .
- 6. Move the touch point over Classes, Parents, or About Us and verify that when you click (tap) the touch point on a submenu title the nested submenu contents are displayed.
 - 7. Verify that you when you click elsewhere in the page the submenu contents are hidden.

Figure 5–16 shows the effect of opening a submenu with the touch emulator.

Figure 5–16 Using the Google Chrome device emulator tool



© Robert Kneschke/Shutterstock.com

- 8. Continue to explore Google Chrome's device emulators, trying out different combinations of devices and screen orientations. Press **F12** again to close the developer window.

An important aspect of mobile design is optimizing your site's performance under varying network conditions. Thus, in addition to emulating the properties of the mobile device, Google Chrome's device emulator can also emulate network connectivity.

Marjorie wants to increase the font size of the links in the navigation list to make them easier to access using touch. She also wants to hide the customer comments that have been placed in the `aside` element (because she doesn't feel this will be of interest to mobile users). Because these changes only apply to the mobile device version of the page, you'll add the style rules within the media query for mobile devices.

To hide the customer comments:

- 1. Return to the `tf_styles1.css` file in your editor and go to the Mobile Styles section.
- 2. Within the media query for screen devices with a maximum width of 480 pixels, add the following style rule to increase the font size of the hypertext links in the navigation list. Indent the style rule to offset it from the braces around the media query.

```
nav.horizontal a {
    font-size: 1.5em;
    line-height: 2.2em;
}
```

The styles rules for a media query must always be placed within curly braces to define the extent of the query.

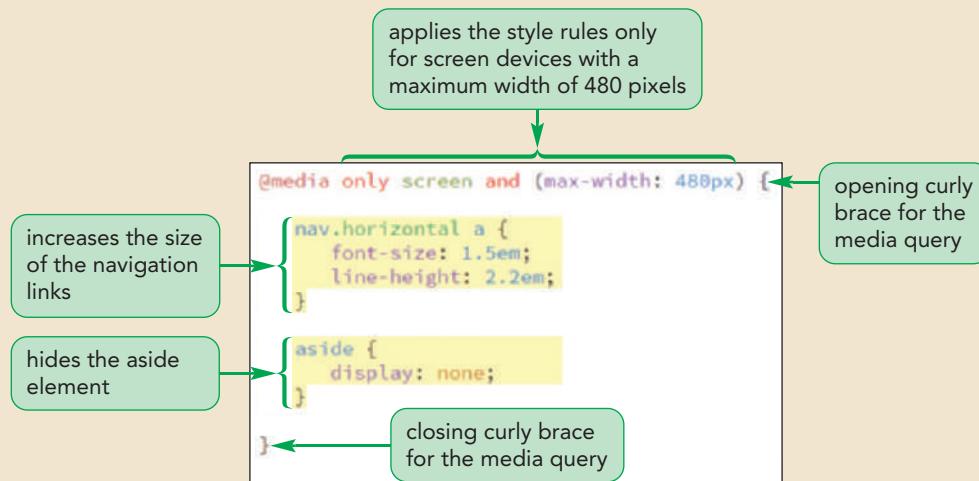
- 3. Add the following style rule to hide the `aside` element (once again indented from the surrounding media query):

```
aside {  
    display: none;  
}
```

Figure 5–17 highlights the style rules in the media query for mobile devices.

Figure 5–17

Hiding the `aside` element for mobile devices

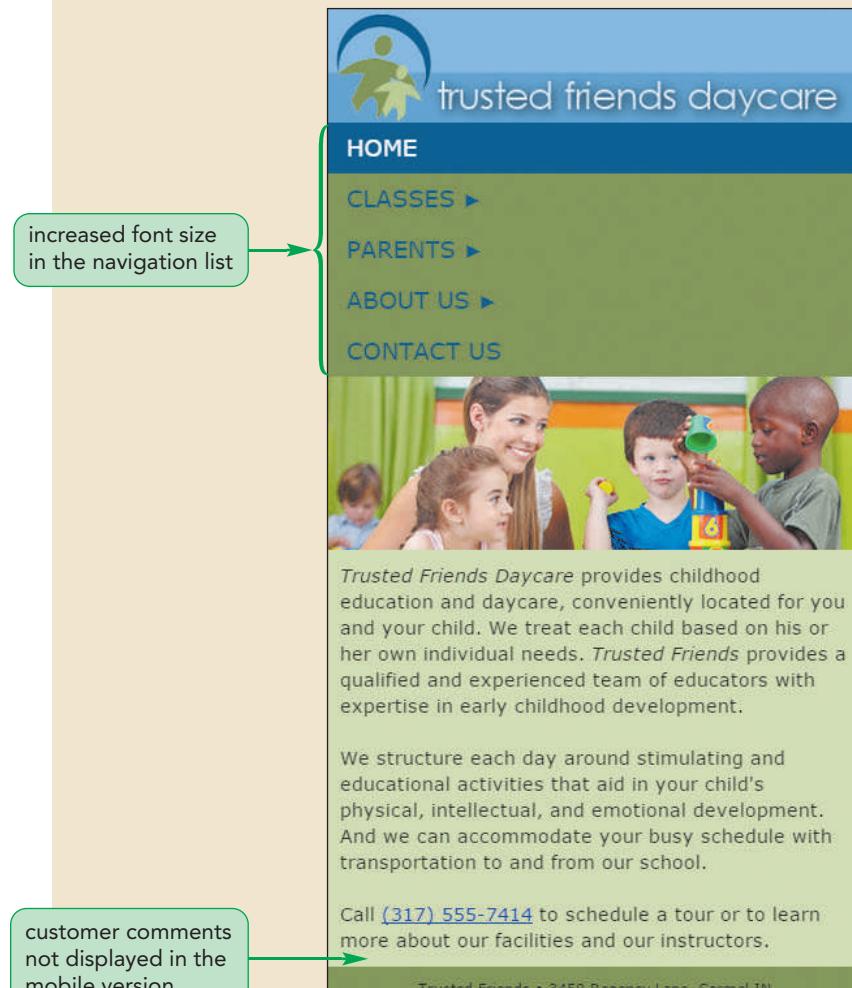


- 4. Save your changes to the file and then reload the `tf_home.html` file in your browser. Reduce the width of the browser window to 480 pixels or below (or view the page in your mobile emulator). Verify that the customer comments are no longer displayed on the web page and that the size of the navigation links has been increased.

Figure 5–18 shows the final design of the mobile version.

Figure 5–18

Final design of the mobile version of the home page



Now that you've completed the mobile design of the page, you'll start to work on the design for tablet devices.

Creating a Tablet Design

Under the media query you've set up, your design for tablet devices will be applied for screen widths greater than 480 pixels. The pulldown menu you created was part of the base styles, so it is already part of the tablet design; however, with the wider screen, Marjorie would like the submenus displayed horizontally rather than vertically. You can accomplish this by adding a style rule to the tablet media query to float the submenus side-by-side.

To begin writing the tablet design:

1. Return to the **tf_styles1.css** file in your editor and scroll down to the media query for the tablet styles.

- 2. Within the media query, add the following style to float the five list items, which are direct children of the main menu, side-by-side. Set the width of each list item to 20% of the total width of the main menu.

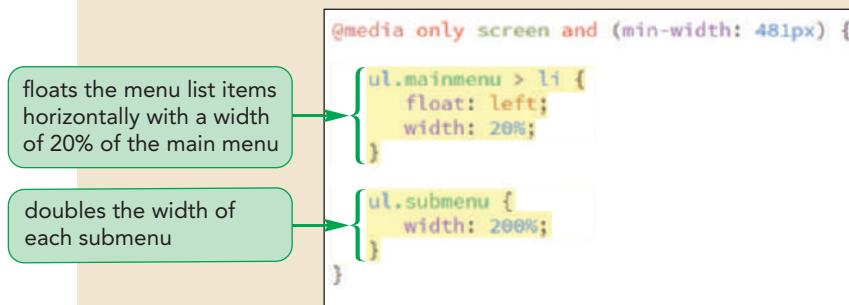
```
ul.mainmenu > li {  
    float: left;  
    width: 20%;  
}
```

- 3. Double the widths of the submenus so that they stand out better from the main menu titles by adding the following style rule.

```
ul.submenu {  
    width: 200%;  
}
```

Figure 5–19 highlights the style rule within the media query for tablet devices.

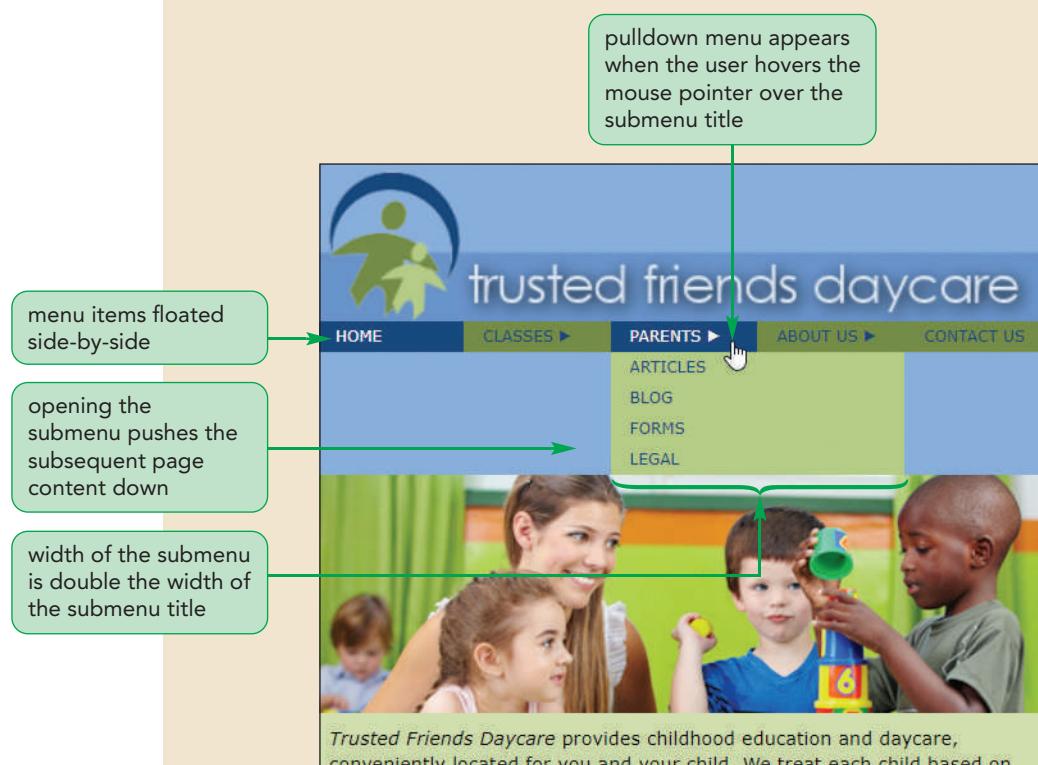
Figure 5–19 **Formatting the navigation menus for tablet devices**



- 4. Save your changes to the style sheet and then reload the `tf_home.html` file in your web browser.
- 5. Increase the width of the browser window beyond 480 pixels to switch from the mobile design to the tablet design. Verify that the submenu titles are now laid out horizontally and that if you hover your mouse pointer over the submenu titles, the contents of the submenu are made visible on the screen. See Figure 5–20.

Figure 5–20

Pulldown menus for the tablet layout



© Robert Kneschke/Shutterstock.com;

- 6. Scroll down as needed and note that the customer comments now appear at the bottom of the page because they were only hidden for the mobile version of this document.

Marjorie notices that opening the submenus pushes the subsequent page content down to make room for the submenu. She prefers the submenus to overlay the page content. You can accomplish this by placing the submenus with absolute positioning. Remember that objects placed with absolute positioning are removed from the document flow and thus, will overlay subsequent page content. To keep the submenus in their current position on the page, you'll make each main list item a container for its submenu by setting its `position` property to `relative`. Thus, each submenu will be placed using absolute positioning with its main list item. You will not need to set the `top` and `left` coordinates for these items because you'll use the default value of 0 for both. Because the submenus will overlay page content, Marjorie suggests you add a drop shadow so, when a submenu is opened, it will stand out more from the page content.

To position the navigation submenus:

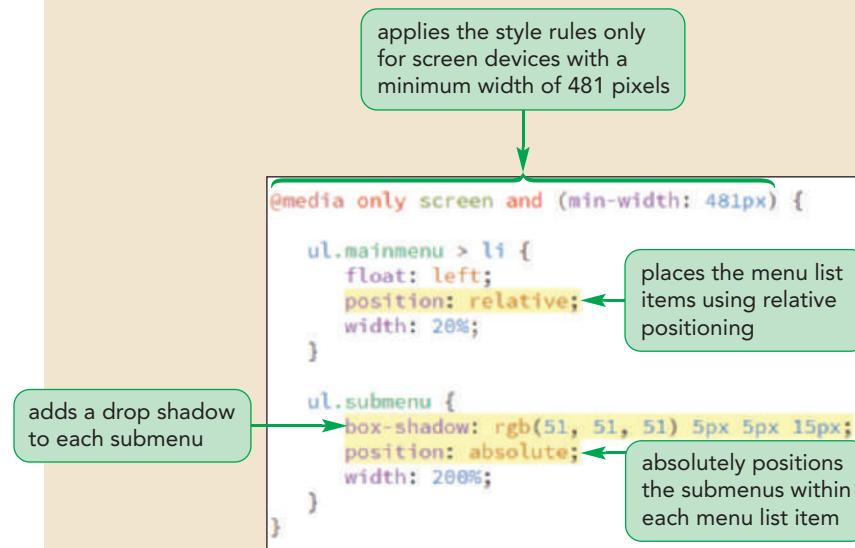
- 1. Return to the `tf_styles1.css` style sheet in your editor.
- 2. Locate the style rule for the `ul.mainmenu > li` selector in the Tablet Styles section and add the following style:
`position: relative;`

- 3. Add the following style to the `ul.submenu` selector in the Tablet Styles section:

```
box-shadow: rgb(51, 51, 51) 5px 5px 15px;
position: absolute;
```

Figure 5–21 highlights the new styles.

Figure 5–21 Placing the pulldown menus with absolute positioning



- 4. Save your changes to the style sheet and then reload the `tf_home.html` file in your web browser.
- 5. Verify that when you open the pulldown menus, the subsequent page content is not shifted downward. Figure 5–22 highlights the final design for the tablet version of the home page.

Figure 5–22 Revised design of the pulldown menus



© Robert Kneschke/Shutterstock.com

You'll complete your work on the home page by creating the desktop version of the page design.

Creating a Desktop Design

Some of the designs that will be used in the desktop version of the page have already been placed in the Base Styles section of the `tf_styles1.css` style sheet. For example, the maximum width of the web page has been set to 1024 pixels. For browser windows that exceed that width, the web page will be displayed on a fixed background image of children playing. Other styles are inherited from the style rules for tablet devices. For example, desktop devices will inherit the style rule that floats the navigation submenus alongside each other within a single row. All of which illustrates an important principle in designing for multiple devices: *don't reinvent the wheel*. As much as possible allow your styles to build upon each other as you move to wider and wider screens.

However, there are some styles that you will have to implement only for desktop devices. With the wider screen desktop screens, you don't need to hide the submenus in a pulldown menu system. Instead you can display all of the links from the navigation list. You'll change the submenu background color to transparent so that it blends in with the navigation list and you'll remove the drop shadows you created for the tablet design. The submenus will always be visible, so you'll change their `display` property from `none` to `block`. Finally, you'll change their position to `relative` because you no longer want to take the submenus out of the document flow and you'll change their width to 100%. Apply the styles now to modify the appearance of the submenus.

To start working on the desktop design:

- 1. Return to the `tf_styles1.css` style sheet in your editor and within the media query for devices with screen widths 769 pixels or greater insert the following style rule to format the appearance of the navigation submenus.

```
ul.submenu {  
    background: transparent;  
    box-shadow: none;  
    display: block;  
    position: relative;  
    width: 100%;  
}
```

- 2. The navigation list itself needs to expand so that it contains all of its floated content. Add the following style rule to the media query for desktop devices:

```
nav.horizontal::after {  
    clear: both;  
    content: "";  
    display: table;  
}
```

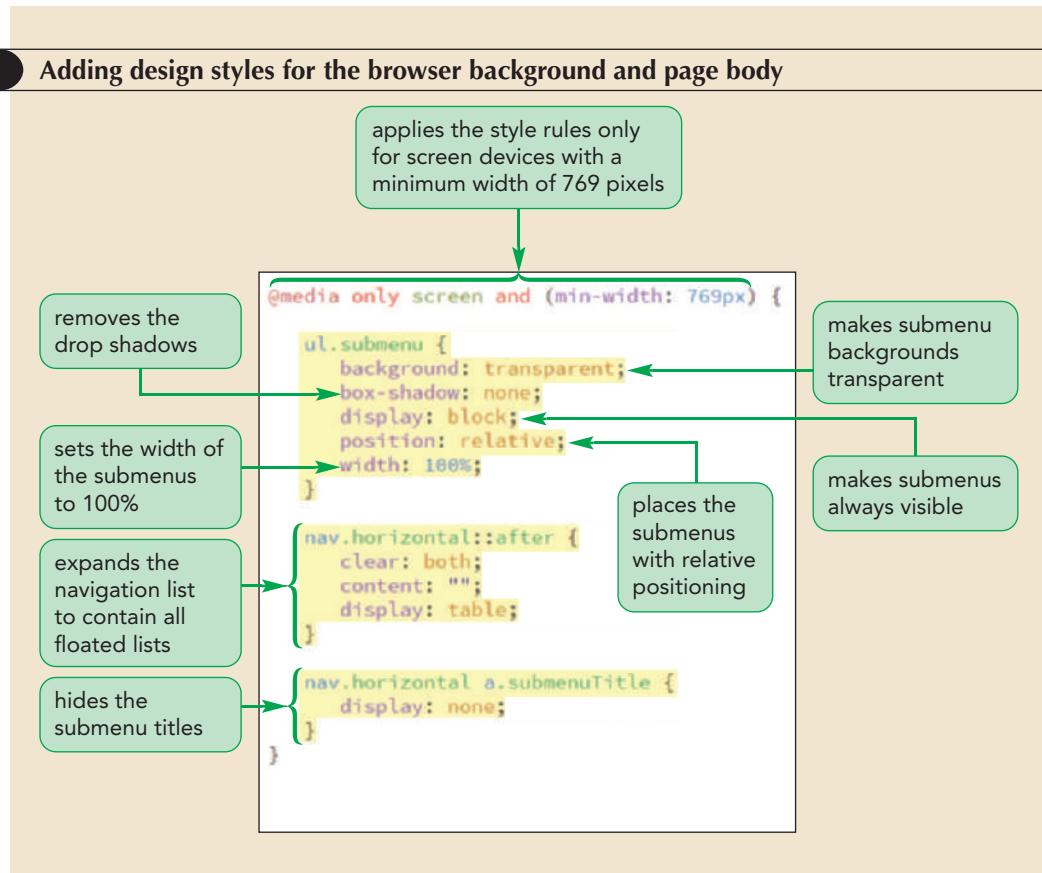
- 3. Finally with no hidden submenus, there is no reason to have a submenu title. Add the following style rule to remove the submenu titles:

```
nav.horizontal a.submenuTitle {  
    display: none;  
}
```

Figure 5–23 highlights the new style rules in the desktop media query.

Figure 5–23

Adding design styles for the browser background and page body



With a wider screen, you want to order to avoid long lines of text, which are difficult to read. Modify the layout of the desktop design so that the main article and the customer comments are floated side-by-side within the same row.

To change the layout of the article and aside elements:

- 1. Within the media query for desktop devices, add the following style rules to float the article and aside elements:

```

article {
    float: left;
    margin-right: 5%;
    width: 55%;
}
aside {
    float: left;
    width: 40%;
}

```

Figure 5–24 highlights the final style rules in the desktop media query.

Figure 5–24

Styles for the article and aside elements

```

nav.horizontal a.submenuTitle {
  display: none;
}

article {
  float: left;
  margin-right: 5%;
  width: 55%;
}

aside {
  float: left;
  width: 40%;
}

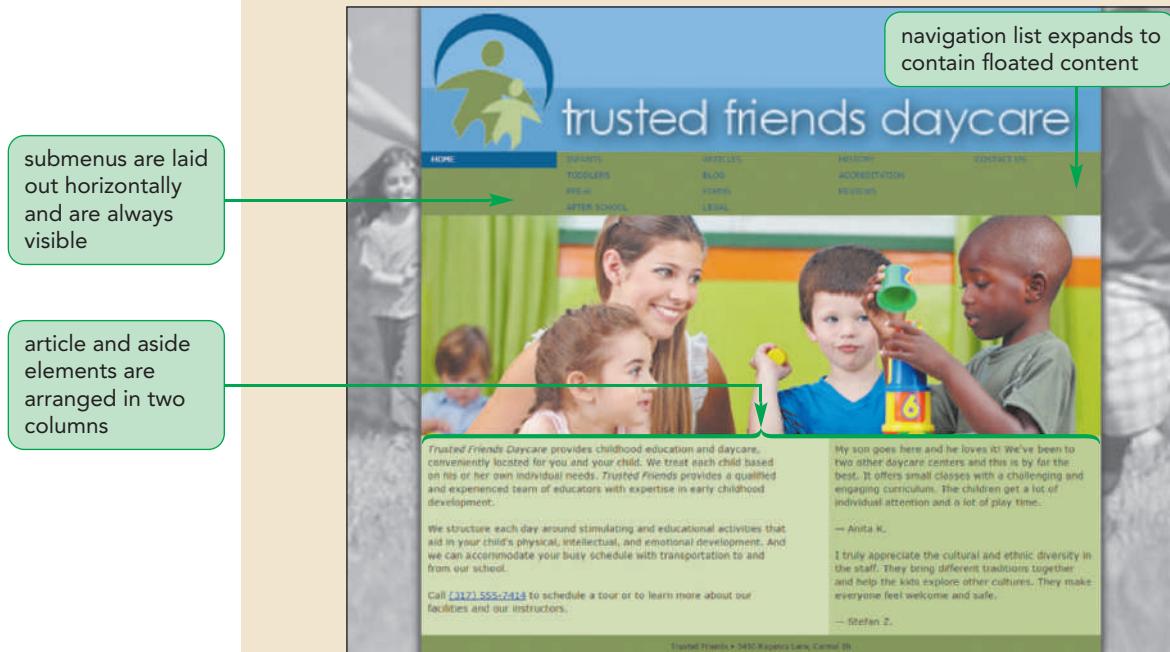
```

- 2. Save your changes to the style sheet and then reload `tf_home.html` in your browser.

Figure 5–25 shows the final appearance of the desktop design.

Figure 5–25

Final desktop design for the Trusted Friends home page



© Robert Kneschke/Shutterstock.com; © dotshock/Shutterstock.com

- 3. Resize your web browser and verify that as you change the browser window width, the layout changes from the mobile to the tablet to the desktop design.

You show the final design of the home page to Marjorie. She is pleased by the changes you've made and likes that the page's content and layout will automatically adapt to different screen widths.



PROSKILLS

Problem Solving: Optimizing Your Site for the Mobile Web

The mobile browser market is a rapidly evolving and growing field with more new devices and apps introduced each month. Adapting your website for the mobile web is not a luxury, but a necessity.

A good mobile design matches the needs of consumers. Mobile users need quick access to main sources of information without a lot of the extra material often found in the desktop versions of their favorite sites. Here are some things to keep in mind as you create your mobile designs:

- *Keep it simple.* To accommodate the smaller screen sizes and slower connection speeds, scale down each page to a few key items and articles. Users are looking for quick and obvious information from their mobile sites.
- *Resize your images.* Downloading several images can bring a mobile device to a crawl. Reduce the number of images in your mobile design, and use a graphics package to resize the images so they are optimized in quality and sized for a smaller screen.
- *Scroll vertically.* Readers can more easily read your page when they only have to scroll vertically. Limit yourself to one column of information in portrait orientation and two columns in landscape.
- *Make your links accessible.* Clicking a small hypertext link is extremely difficult to do on a mobile device with a touch screen interface. Create hypertext links that are easy to locate and activate.

Above all, test your site on a variety of devices and under different conditions. Mobile devices vary greatly in size, shape, and capability. What works on one device might fail utterly on another. Testing your code on a desktop computer is only the first step; you may also need access to the devices themselves. Even emulators cannot always capture the nuances involved in the performance of an actual mobile device.

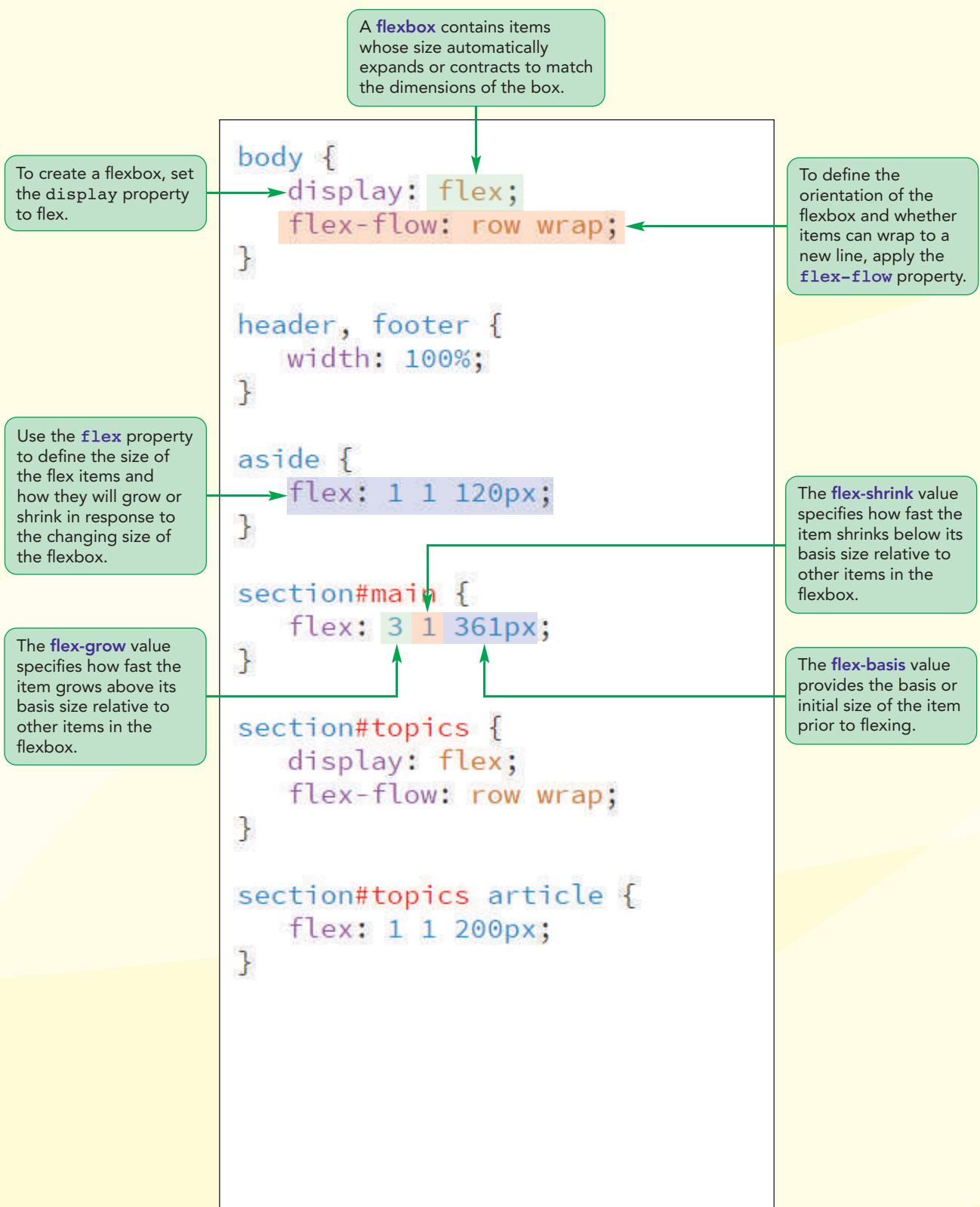
You've completed your work on the design of the Trusted Friends home page with a style sheet that seamlessly transitions between mobile, tablet, and desktop devices. In the next session, you'll explore how to use flexible boxes to achieve a responsive design.

REVIEW

Session 5.1 Quick Check

1. Which of the following is *not* a part of responsive design theory?
 - a. flexible layouts
 - b. pulldown menus
 - c. image rescaling
 - d. media queries
2. Which attribute do you add to a `link` element for aural browsers?
 - a. `media = "aural"`
 - b. `type = "aural"`
 - c. `media = "speech"`
 - d. `type = "speech"`
3. What `@rule` do you use for braille device?
 - a. `@media braille`
 - b. `@braille true`
 - c. `@type braille`
 - d. `@media nonscreen`
4. What `@rule` loads style rules for screen devices up to a maximum width of 780 pixels?
 - a. `@screen: 780px`
 - b. `@media screen and (width: 780px)`
 - c. `@screen and (width <= 780px)`
 - d. `@media screen and (max-width: 780px)`
5. What attribute would you add to a `link` element for screen devices whose width ranges from 480 pixels up to 780 pixels (inclusive)?
 - a. `media="screen" min-width="480px" min-width="480px"`
 - b. `media="screen and (width=480px - 780px)"`
 - c. `minScreenWidth = "480px" maxScreenWidth = "780px"`
 - d. `media="screen and (min-width: 480px and max-width: 780px)"`
6. In general, what media rules should be listed first in your media queries if you want to support mobile, tablet, laptop, and desktop devices?
 - a. mobile
 - b. tablet
 - c. laptop
 - d. desktop
7. Which viewport displays the web page content that fits within mobile screen?
 - a. layout
 - b. visual
 - c. webpage
 - d. browser
8. Which viewport contains the entire content of the page, some of which may be hidden from the user?
 - a. layout
 - b. visual
 - c. webpage
 - d. browser

Session 5.2 Visual Overview:



© dotshock/Shutterstock.com; BenBois/openclipart; JMLevick/openclipart

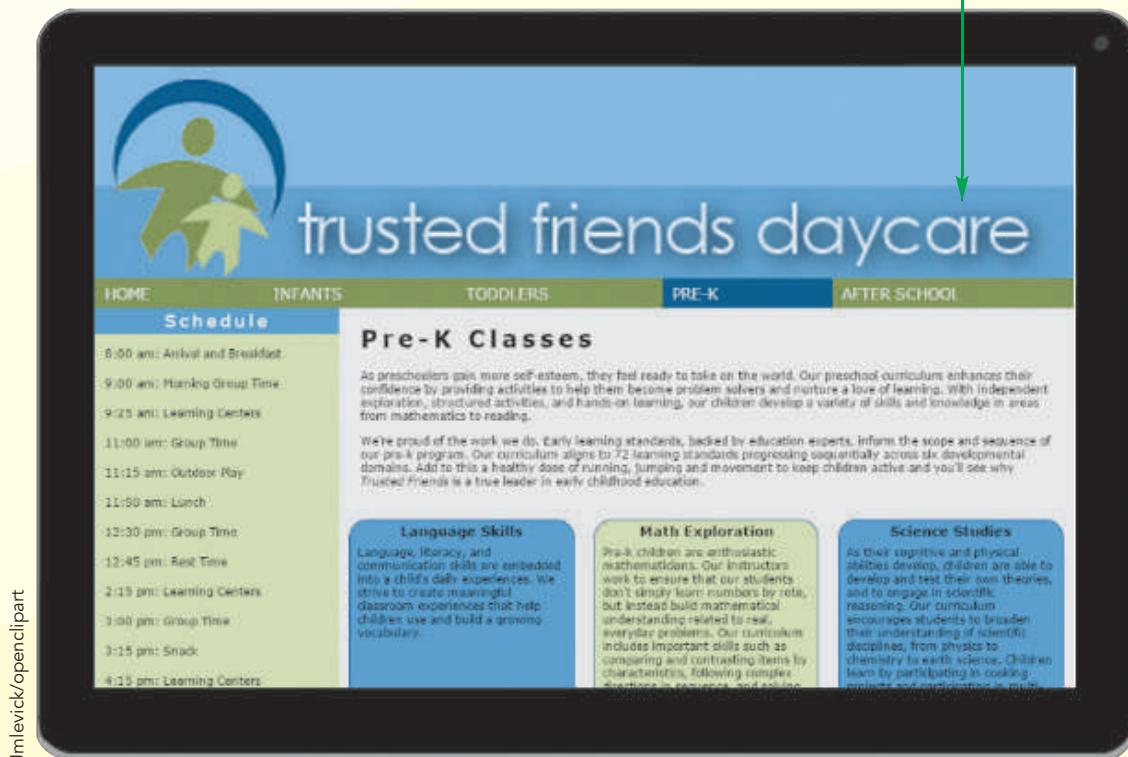
Flexible Layouts



BenBois/openclipart

With narrower screens, a flexbox layout automatically places items within a single column.

With wider screens, the items are free to expand, automatically placing themselves into multiple columns.



Jmlevick/openclipart

Copyright 2021 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s). Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Introducing Flexible Boxes

So far our layouts have been limited to a grid system involving floating elements contained within a fixed or fluid grid of rows and columns. One of the challenges of this approach under responsive design is that you need to establish a different grid layout for each class of screen size. It would be much easier to have a single specification that automatically adapts itself to the screen width without requiring a new layout design. One way of achieving this is with flexible boxes.

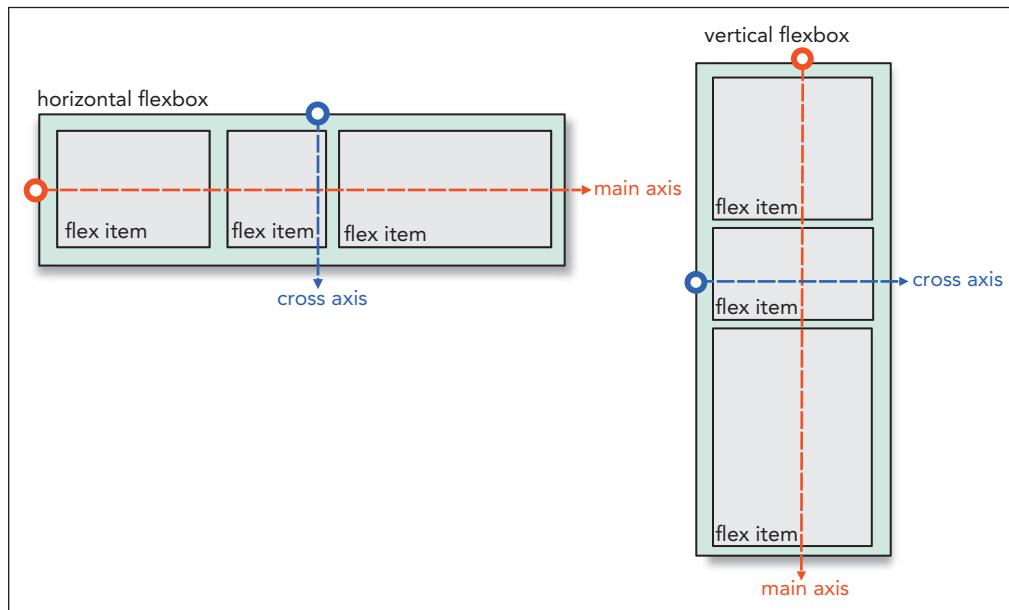
Defining a Flexible Box

A flexible box or flexbox is a box containing items whose sizes can shrink or grow to match the boundaries of the box. Thus, unlike a grid system in which each item has a defined size, flexbox items adapt themselves automatically to the size of their container. This makes flexboxes a useful tool for designing layouts that can adapt to different page sizes.

Items within a flexbox are laid out along a **main axis**, which can point in either the horizontal or vertical direction. Perpendicular to the main axis is the **cross axis**, which is used to define the height or width of each item. Figure 5–26 displays a diagram of two flexboxes with items arranged either horizontally or vertically along the main axis.

Figure 5–26

Horizontal and vertical flexboxes



To define an element as a flexbox, apply either of the following `display` styles

`display: flex;`

or

`display: inline-flex;`

where a value of `flex` starts the flexbox on a new line (much as a block element starts on a new line) and a value of `inline-flex` keeps the flexbox in-line with its surrounding content.

Cross-Browser Flexboxes

The syntax for flexboxes has gone through major revisions as it has developed from the earliest drafts to the latest specifications. Many older browsers employ a different flexbox syntax, in some cases replacing the word *flex* with *box* or *flexbox*. The complete list of browser extensions that define a flexbox would be entered as:

```
display: -webkit-box;
display: -moz-box;
display: -ms-flexbox;
display: -webkit-flex;
display: flex;
```

To simplify the code in the examples that follow, you will limit your code to the W3C specification. This will cover the current browsers at the time of this writing. However, if you need to support older browsers, you may have to include a long list of browser extensions for each flex property.

Setting the Flexbox Flow

By default, flexbox items are arranged horizontally starting from the left and moving to the right. To change the orientation of the flexbox, apply the following `flex-direction` property

```
flex-direction: direction;
```

where `direction` is `row` (the default), `column`, `row-reverse`, or `column-reverse`. The `row` option lays out the flex items from left to right, `column` creates a vertical layout starting from the top and moving downward, and the `row-reverse` and `column-reverse` options lay out the items bottom-to-top and right-to-left respectively.

Flex items will all try to fit within a single line, either horizontally or vertically. But if they can't, those items can wrap to a new line as needed by applying the following `flex-wrap` property to the flexbox

```
flex-wrap: type;
```

where `type` is either `nowrap` (the default), `wrap` to `wrap` the flex items to a new line, or `wrap-reverse` to wrap flex items to a new line starting in the opposite direction from the current line. For example, the following style rules create a flexbox in which the items are arranged in a column starting from the top and going down with any flex items that wrap to the second column starting from the bottom and moving up.

```
display: flex;
flex-direction: column;
flex-wrap: wrap-reverse;
```

TIP

Some older browsers do not support the `flex-flow` property, so for full cross-browser support, you might use the `flex-direction` and `flex-wrap` properties instead.

Additional items in this flexbox will continue to follow a snake-like curve with the third column starting at the top, moving down, and so forth.

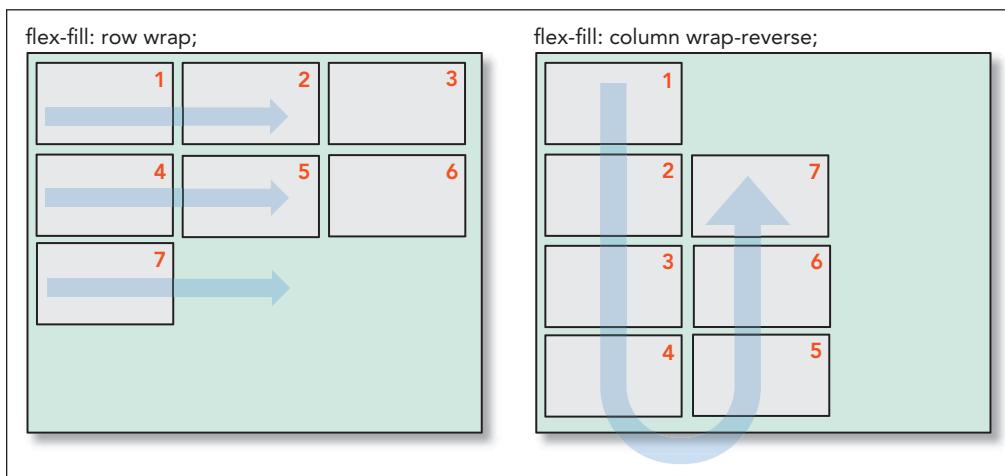
Both the `flex-direction` and `flex-wrap` properties can be combined into the following `flex-flow` style

```
flex-flow: direction wrap;
```

where `direction` is the direction of the flex items and `wrap` defines whether the items will be wrapped to a new line when needed. Figure 5-27 shows an example of flexboxes laid out in rows and columns in which the flex items are forced to wrap to a new line. Note that the column-oriented flexbox uses `wrap-reverse` to start the new column on the bottom rather than the top.

Figure 5–27

Flexbox layouts



REFERENCE

Defining a Flexbox

- To display an element as a flexbox, apply the `display` style
`display: flex;`
- To set the orientation of the flexbox, apply the style
`flex-direction: direction;`
 where `direction` is `row` (the default), `column`, `row-reverse`, or `column-reverse`.
- To define whether or not flex items wrap to a new line, apply the style
`flex-wrap: type;`
 where `type` is either `nowrap` (the default), `wrap` to wrap flex items to a new line, or `wrap-reverse` to wrap flex items to a new line starting in the opposite direction from the current line.
- To define the flow of items within a flexbox, apply the style
`flex-flow: direction wrap;`
 where `direction` is the direction of the flex items and `wrap` defines whether the items will be wrapped to a new line when needed.

Marjorie wants you to use flexboxes to design a page she's created describing the pre-k classes offered by Trusted Friends. She has already created the content of the page and several style sheets to format the appearance of the page elements. You'll create a style sheet that lays out the page content drawing from a library of flexbox styles.

To open the pre-k page and style sheet:

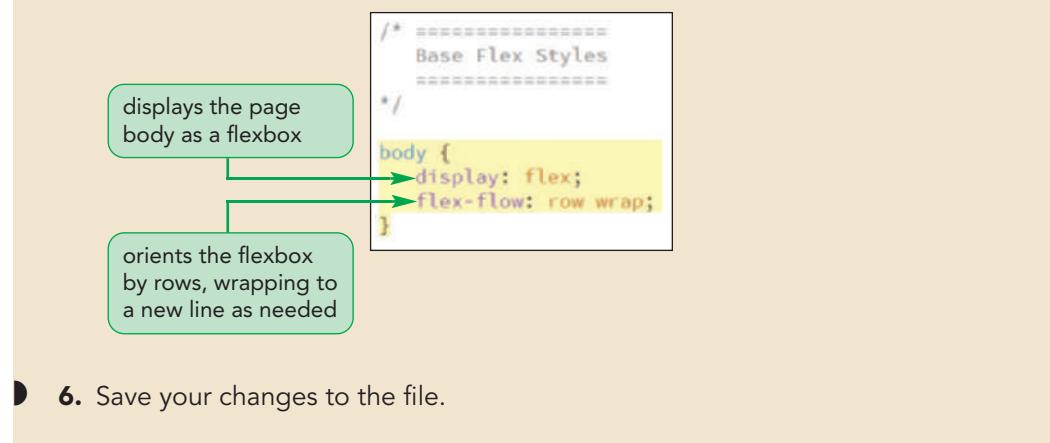
- 1. Use your editor to open the `tf_prek_txt.html` and `tf_flex_txt.css` files from the `html05` ▶ `tutorial` folder. Enter **your name** and **the date** in the comment section of each file and save them as `tf_prek.html` and `tf_flex.css` respectively.
- 2. Return to the `tf_prek.html` file in your editor and, within the document head, create links to the `tf_reset.css`, `tf_styles2.css`, and `tf_flex.css` style sheets in that order.

- 3. Take some time to scroll through the contents of the document to become familiar with its contents and structure and then save your changes to the file, leaving it open.
- 4. Go to the **tf_flex.css** file in your editor.
- 5. Go to the Base Flex Styles section and insert the following style rules to display the entire page body as a flexbox oriented horizontally with overflow flex items wrapped to a new row as needed:

```
body {
  display: flex;
  flex-flow: row wrap;
}
```

Figure 5–28 highlights the new flexbox styles in the style sheet.

Figure 5–28 Setting the flex display style



- 6. Save your changes to the file.

Now that you've defined the page body as a flexbox, you'll work with styles that define how items within a flexbox expand and contract to match the flexbox container.

Working with Flex Items

Flex items behave a lot like floated objects though with several advantages, including that you can float them in either the horizontal or vertical direction and that you can change the order in which they are displayed. While the size of a flex item can be fixed using the CSS `width` and `height` properties, they don't have to be. They can also be "flexed"—automatically adapting their size to fill the flexbox. A flex layout is fundamentally different from a grid layout and requires you to think about sizes and layout in a new way.

TIP

Because flexboxes can be aligned horizontally or vertically, the `flex-basis` property sets either the initial width or the initial height of the flex item depending on the orientation of the flexbox.

Setting the Flex Basis

When items are allowed to "flex" their rendered size is determined by three properties: the basis size, the growth value, and the shrink value. The basis size defines the initial size of the item before the browser attempts to fit it to the flexbox and is set using the following `flex-basis` property

```
flex-basis: size;
```

where `size` is one of the CSS units of measurement, a percentage of the size of the flexbox, or the keyword `auto` (the default), which sets the initial size of the flex item based on its content or the value of its `width` or `height` property. For example, the following style rule sets the initial size of the `aside` element to 200 pixels:

```
aside {  
    flex-basis: 200px;  
}
```

The `flex-basis` property should not be equated with the `width` and `height` properties used with grid layouts; rather, it serves only as a starting point. The actual rendered size of the `aside` element in this example is not necessarily 200 pixels but will be based on the size of the flexbox, as well as the size of the other items within the flexbox.

Defining the Flex Growth

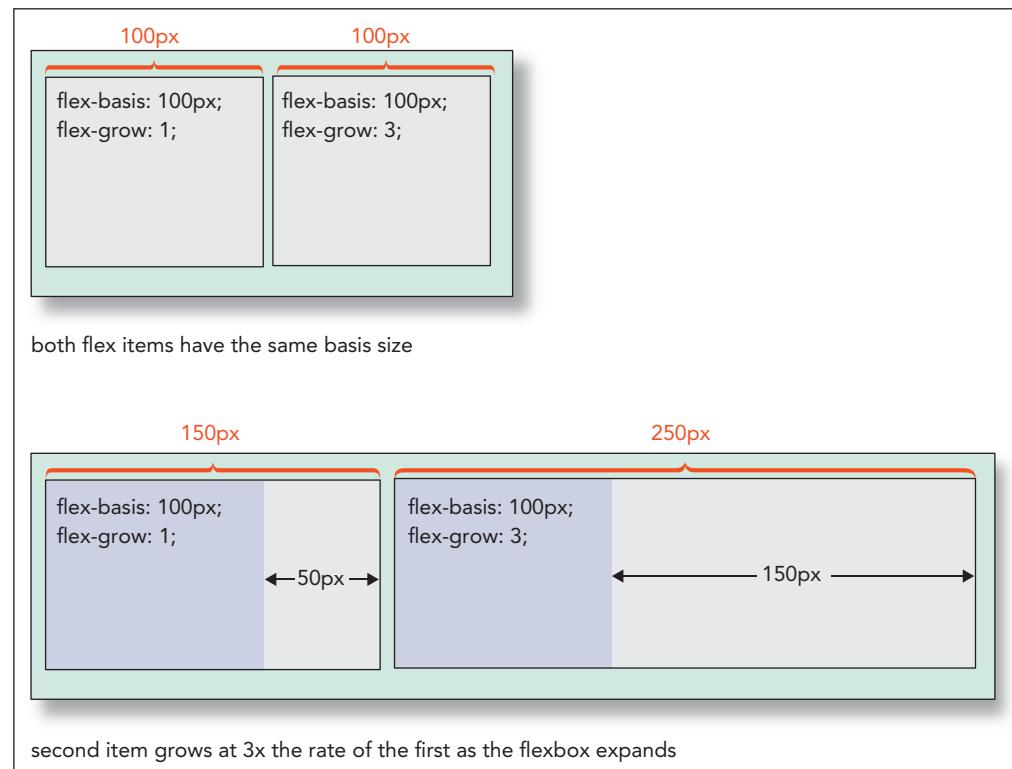
Once the basis size of the item has been defined, the browser will attempt to expand the item into its flexbox. The rate at which a flex item grows from its basis size is determined by the following `flex-grow` property

flex-grow: value;

where `value` is a non-negative value that expresses the growth of the flex item relative to the growth of the other items in the flexbox. The default `flex-grow` value is 0, which is equivalent to not allowing the flex item to grow but to remain at its basis size. Different items within a flexbox can have different growth rates and the growth rate largely determines how much of the flexbox is ultimately occupied by each item.

Figure 5–29 shows an example of how changing the size of a flexbox alters the size of the individual flexbox items.

Figure 5–29 Growing flex items beyond their basis size



In the figure, the basis sizes of the two items are 100 pixels each with the growth of the first item set to 1 and the growth of the second item set to 3. The growth values indicate that as the flex items expand to fill the flexbox, item1 will increase 1 pixel for every 3 pixels that item2 increases. Thus, to fill up the remaining 200 pixels of a 400-pixel wide flexbox, 50 pixels will be allotted to the first item and 150 pixels will be allotted to the second item, resulting in final sizes of 150 pixels and 250 pixels respectively. If the width of the flexbox were to increase to 600 pixels, item1 and item2 will divide the extra 400 pixels once again in a ratio of 1 to 3. Item1 will have a total size of 200 pixels (100px + 100px) and item2 will expand to a size of 400 pixels (100px + 300px).

TIP

If all items have `flex-grow` set to 1 and an equal `flex-basis`, they will always have an equal size within the flexbox.

Notice that unlike a grid layout, the relative proportions of the items under a flex layout need not be constant. For the layout shown in Figure 5–29, the two items share the space equally when the flexbox is 200 pixels wide, but at 400 pixels the first item occupies 37.5% of the box while the second item occupies the remaining 62.5%.

To keep a constant ratio between the sizes of the flex items, set their basis sizes to 0 pixels. For example, the following style rules will result in a flexbox in which the first item is always half the size of the second item no matter how wide or tall the flexbox becomes.

```
div#item1 {
  flex-basis: 0px;
  flex-grow: 1;
}
div#item2 {
  flex-basis: 0px;
  flex-grow: 2;
}
```

One of the great advantages of the flexible box layout is that you don't need to know how many items are in the flexbox to keep their relative proportions the same. The following style rule creates a layout for a navigation list in which each list item is assigned an equal size and grows at the same rate.

```
nav ul {
  display: flex;
}
nav ul li {
  flex-basis: 0px;
  flex-grow: 1;
}
```

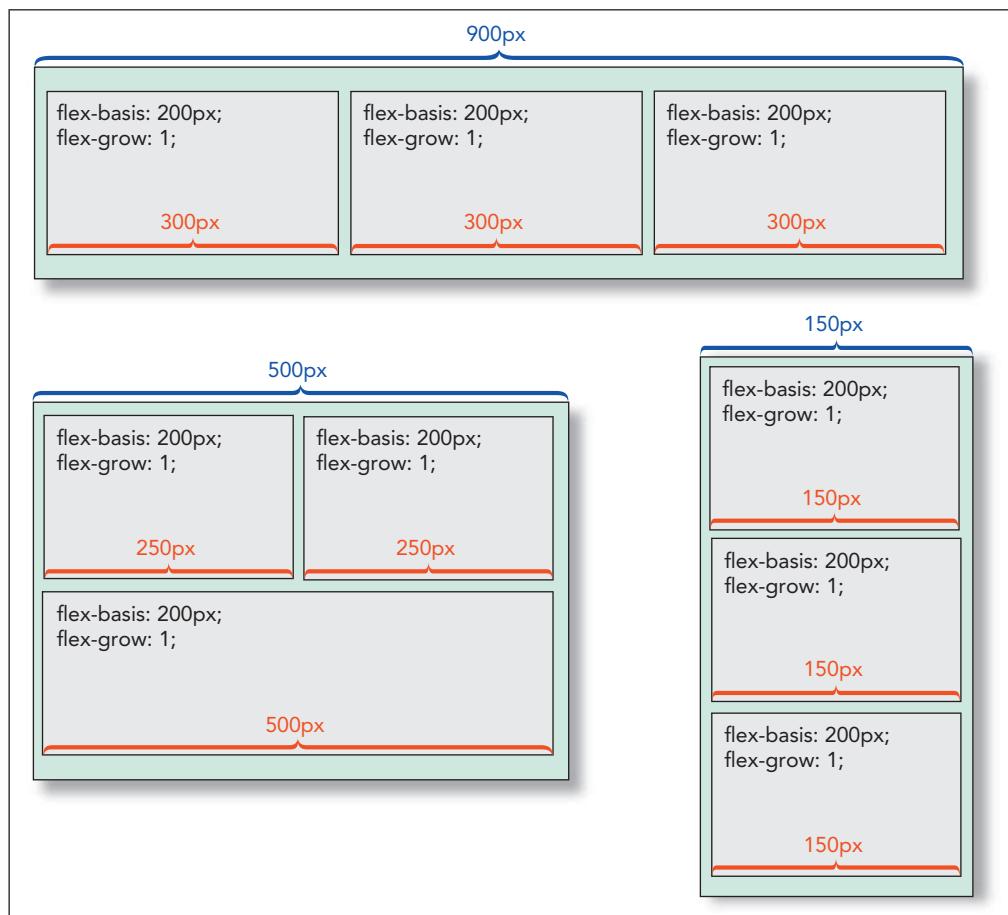
If there are four items in this navigation list, each will be 25% of the total list size and if at a later date a fifth item is added, those items will then be allotted 20% of the total size. Thus, unlike a grid layout, there is no need to revise the percentages to accommodate new entries in the navigation list; a flexible box layout handles that task automatically.

Note that if the `flex-grow` value is set to 0, the flex item will not expand beyond its basis size, making that basis value the maximum width or height of the item.

Defining the Shrink Rate

What happens when the flexbox size falls below the total space allotted to its flex items? There are two possibilities depending on whether the flexbox is defined to wrap its contents to a new line. If the `flexbox-wrap` property is set to `wrap`, one or more of the flex items will be shifted to a new line and expanded to fill in the available space on that line. Figure 5–30 shows a flexbox layout in which three items each have a basis size of 200 pixels with the same growth value of 1.

Figure 5–30 Shrinking flex items smaller than their basis size



As shown in the figure, as long as the flexbox is at least 600 pixels wide, the items will equally share a single row. However, once the flexbox size falls below 600 pixels, the three items can no longer share that row and the last item is wrapped to a new row. Once on that new row, it's free to fill up the available space while the first two items equally share the space on the first row. As the flexbox continues to contract, falling below 400 pixels, the first two items can no longer share a row and the second item now wraps to its own row. At this point the three items fill separate rows and as the flexbox continues to shrink, their sizes also shrink.

If the flexbox doesn't wrap to a new line as it is resized, then the flex items will continue to shrink, still sharing the same row or column. The rate at which they shrink below their basis size is given by the following `flex-shrink` property

```
flex-shrink: value;
```

where `value` is a non-negative value that expresses the shrink rate of the flex item relative to the shrinkage of the other items in the flexbox. The default `flex-shrink` value is 1. For example, in the following style rules, item1 and item2 will share the flexbox equally as long as the width of the flexbox is 400 pixels or greater.

```
div {
  display: flex;
  flex-wrap: nowrap;
}
div #item1 {
  flex-basis: 200px;
  flex-grow: 1;
  flex-shrink: 3;
}
```

```
div #item2 {
  flex-basis: 200px;
  flex-grow: 1;
  flex-shrink: 1;
}
```

However, once the flexbox falls below 400 pixels, the two items begin to shrink with item1 losing 3 pixels for every 1 pixel lost by item2. Note that if the `flex-shrink` value is set to 0, then the flex item will not shrink below its basis value, making that basis value the minimum width or height of the item.

The `flex` Property

All of the size values described above are usually combined into the following `flex` property

```
flex: grow shrink basis;
```

where `grow` defines the growth of the flex item, `shrink` provides its shrink rate, and `basis` sets the item's initial size. The default `flex` value is

```
flex: 0 1 auto;
```

which automatically sets the size of the flex item to match its content or the value of its `width` and `height` property. The flex item will not grow beyond that size but, if necessary, it will shrink as the flexbox contracts.

The `flex` property supports the following keywords:

- `auto` Use to automatically resize the item from its default size (equivalent to `flex: 1 1 auto;`)
- `initial` The default value (equivalent to `flex: 0 1 auto;`)
- `none` Use to create an inflexible item that will not grow or shrink (equivalent to `flex: 0 0 auto;`)
- `inherit` Use to inherit the flex values of its parent element

As with other parts of the flex layout model, the `flex` property has gone through several syntax changes on its way to its final specification. To support older browsers, use the browser extensions: `-webkit-box`, `-moz-box`, `-ms-flexbox`, `-webkit-flex`, and `flex` in that order.

Sizing Flex Items

- To set the initial size of a flex item, apply the style

```
flex-basis: size;
```

where `size` is measured in one of the CSS units of measurement or as a percentage of the size of the flexbox or the keyword `auto` (the default).

- To define the rate at which a flex item grows from its basis size, apply the style

```
flex-grow: value;
```

where `value` is a non-negative value that expresses the growth of the flex item relative to the growth of the other items in the flexbox (the default is 0).

- To define the rate at which a flex item shrinks below its basis value, apply

```
flex-shrink: value;
```

where `value` is a non-negative value that expresses the shrink rate of the flex item relative to other items in the flexbox (the default is 0).

- To define the overall resizing of a flex item, apply

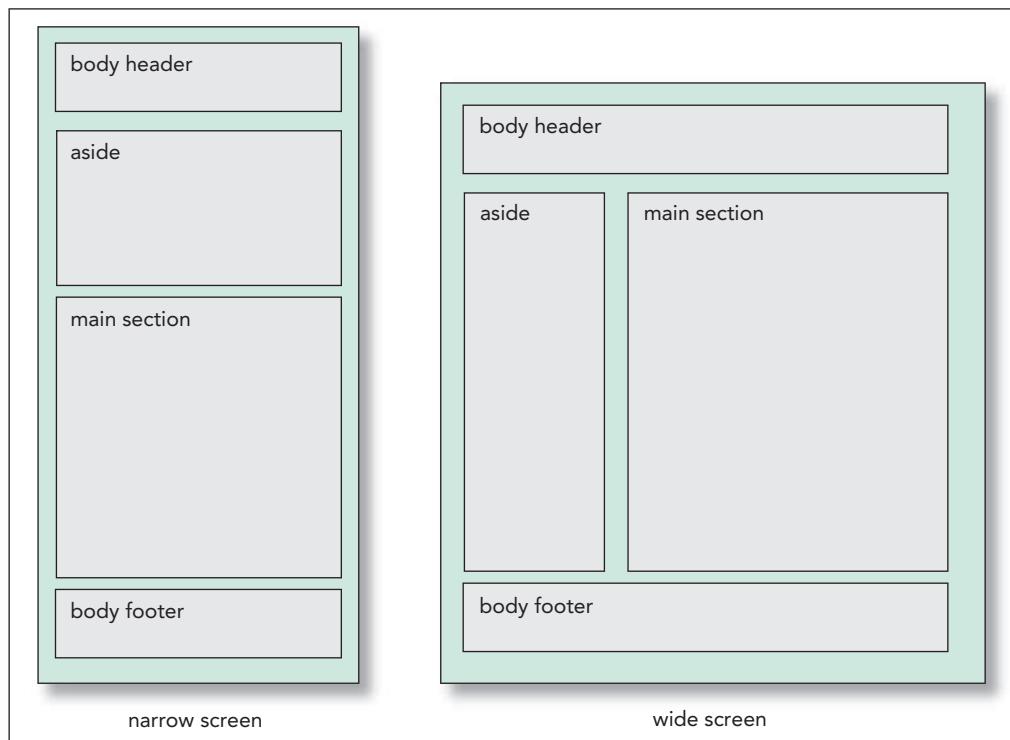
```
flex: grow shrink basis;
```

where `grow` defines the growth of the flex item, `shrink` provides its shrink rate, and `basis` sets the item's initial size.

Applying a Flexbox Layout

Now that you've seen how to size items within a flexbox, you can return to the layout for the Pre-K Classes page at Trusted Friends Daycare. The `body` element, which you already set up as a flexbox, has four child elements: the page header, an `aside` element describing the daily class schedule, a `section` element describing the classes, and the page footer. Marjorie wants the header and the footer to always occupy a single row at 100% of the width of the page body. For wide screens, she wants the `aside` and `section` elements displayed side-by-side with one-fourth of the width assigned to the `aside` element and three-fourths to the `section` element. For narrow screens, she wants the `aside` and `section` elements displayed within a single column. Figure 5–31 displays the flex layout that Marjorie wants you to apply.

Figure 5–31 Proposed flex layout for the Pre-K page



Using the techniques of the first session, this would require media queries with one grid layout for narrow screens and a second grid layout for wide screens. However, you can accomplish the same effect with a single flex layout. First, you set the width of the body header and footer to 100% because they will always occupy their own row:

```
header, footer {
    width: 100%;
}
```

Then, you set the basis size of the `aside` and `section` elements to 120 and 361 pixels respectively. As long as the screen width is 481 pixels or greater, these two elements will be displayed side-by-side; however, once the screen width drops below 481 pixels, the elements will wrap to separate rows as illustrated in the narrow screen image in Figure 5–31. Because you want the `main section` element to grow at a rate three times faster than the `aside` element (in order to maintain the 3:1 ratio in their sizes), you set the `flex-growth` values to 1 and 3 respectively. The flex style rules are

```
aside {
    flex: 1 1 120px;
}
```

```
section#main {
    flex: 3 1 361px;
}
```

Note that you choose 481 pixels as the total initial size of the two elements to match the cutoff point in the media query between mobile and tablet/desktop devices. Generally, you want your flex items to follow the media query cutoffs whenever possible. Add these style rules to the `tf_flex.css` style sheet now.

To define the flex layout:

- 1. Within the `tf_flex.css` file in your editor, add the following style rules to the Base Flex Styles section:

```
header, footer {
    width: 100%;
}

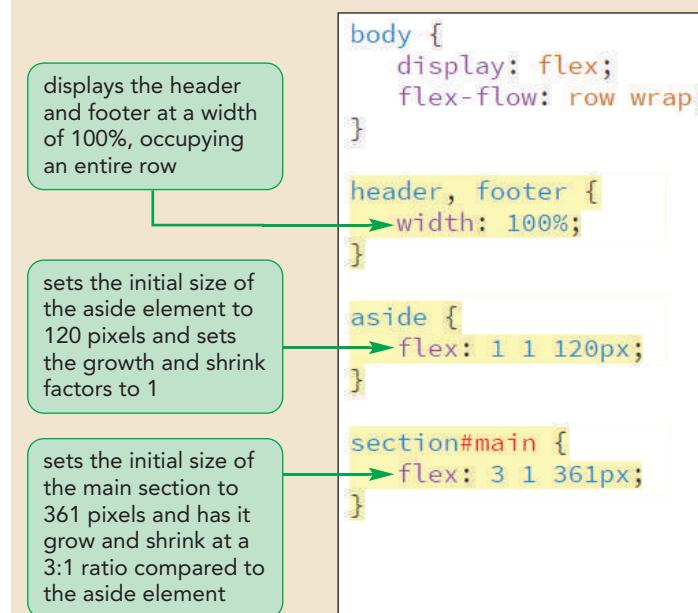
aside {
    flex: 1 1 120px;
}

section#main {
    flex: 3 1 361px;
}
```

Figure 5–32 highlights the newly added style rules to define the flex item sizes.

Figure 5–32

Set the flex properties of the flex items in the page body



- 2. Save your changes to the file and then open the `tf_prek.html` file in your web browser.
- 3. Change the size of the browser window or use the device emulator tools in your browser to view the page under different screen widths. As shown in Figure 5–33, the layout of the page changes as the screen narrows and widens.

Figure 5-33

Flex layout under different screen widths



Flexboxes can be nested within one another and a flex item can itself be a flexbox for its child elements. Within the topics section, Marjorie has created six articles describing different features of the center's pre-k curriculum. She wants these articles to share equal space within a row-oriented flexbox, with each article given a basis size of 200 pixels. The style rules are:

```
section#topics {
    display: flex;
    flex-flow: row wrap;
}

section#topics article {
    flex: 1 1 200px;
}
```

Marjorie also wants the items in the navigation list to appear in a row-oriented flexbox for tablet and desktop devices by adding the following style rules to the media query for screen devices whose width exceeds 480 pixels:

```
nav.horizontal ul {
    display: flex;
    flex-flow: row nowrap;
}

nav.horizontal li {
    flex: 1 1 auto;
}
```

The navigation list items will appear in a single row with no wrapping and the width of each item will be determined by the item's content so that longer entries are given more horizontal space. With the growth and shrink values set to 1, each list item will grow and shrink at the same rate, keeping the layout consistent across different screen widths.

Add these style rules now.

To lay out the topic articles and navigation list:

- 1. Return to the **tf_flex.css** file in your editor and go to the Base Flex Styles section.
- 2. Add the following style rules to create a flex layout for the page articles.

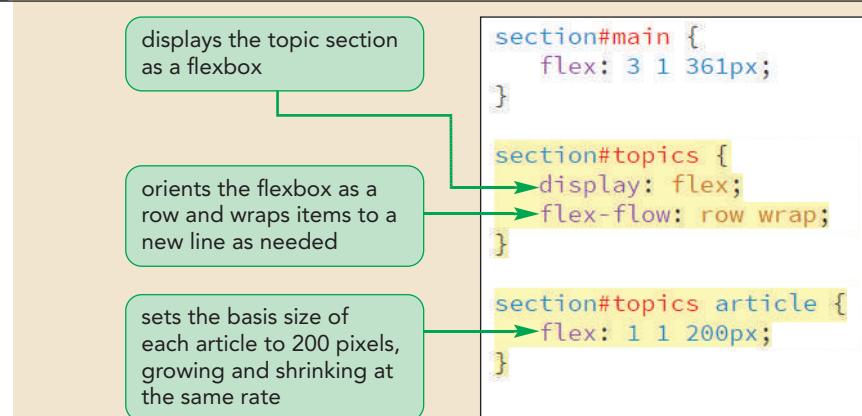
```
section#topics {
  display: flex;
  flex-flow: row wrap;
}

section#topics article {
  flex: 1 1 200px;
}
```

Figure 5–34 highlights the style rules for the article topics layout.

Figure 5–34

Creating a flex layout for articles in the topics section



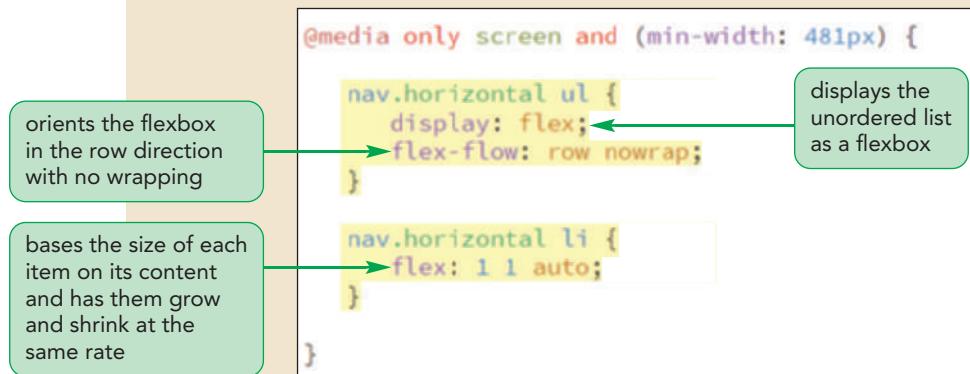
- 3. Scroll down to the media query for tablet and desktop devices and add the following style rule to create a flex layout for the navigation list. (Indent your code to set it off from the media query braces.)

```
nav.horizontal ul {
  display: flex;
  flex-flow: row nowrap;
}

nav.horizontal li {
  flex: 1 1 auto;
}
```

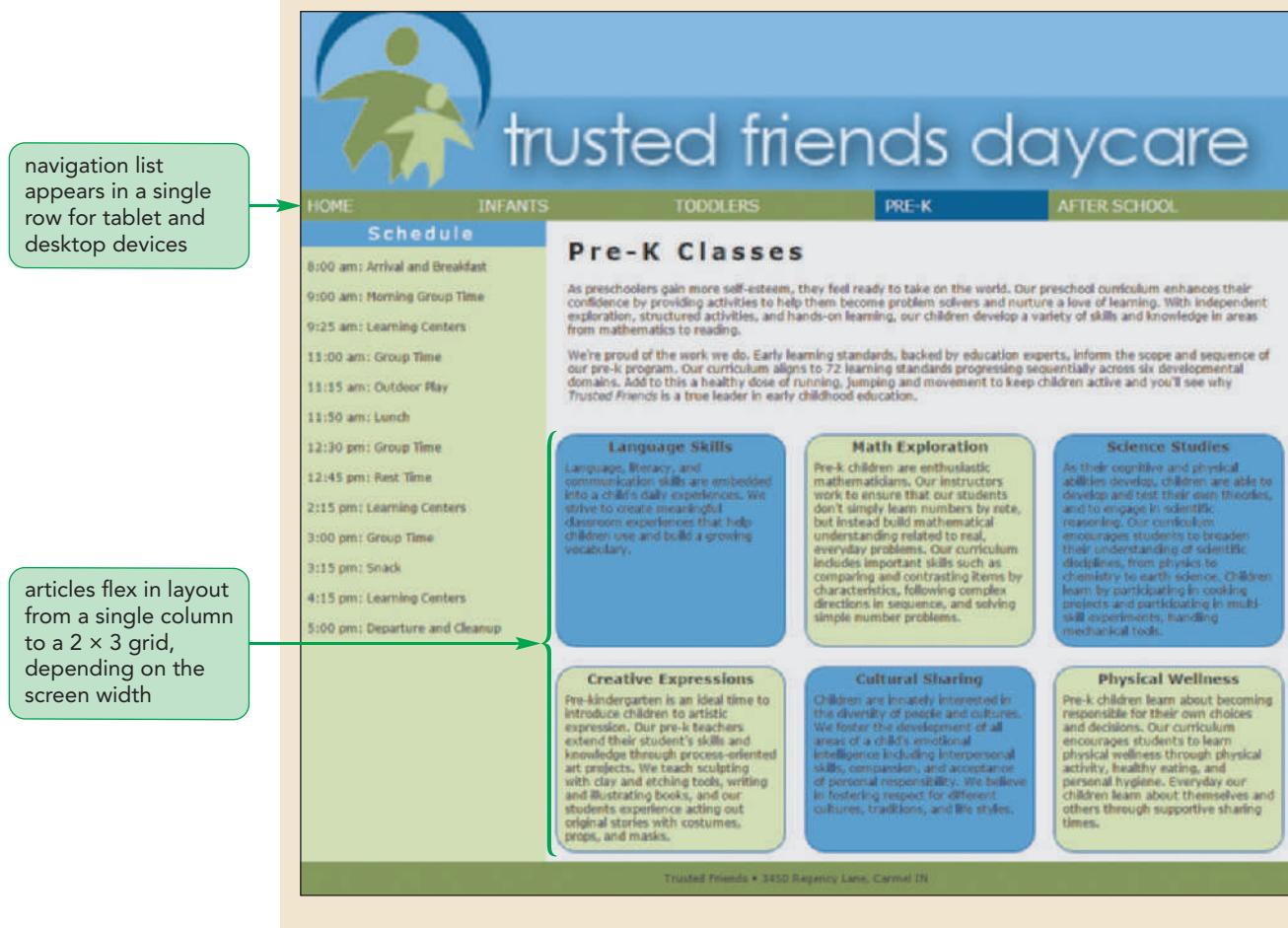
Figure 5–35 highlights the style rules for the navigation list and list items.

Figure 5–35 Creating a flex layout for the navigation list



- 4. Save your changes to the file and reload the `tf_prek.html` file in your web browser.
- 5. View the page under different screen widths and verify that, for tablet and desktop screen widths, the navigation list entries appear in a single row. Also, verify that the articles in the topics section flex from a single column layout to two or more rows of content. See Figure 5–36.

Figure 5–36 Flex layout under a desktop screen width



Marjorie likes how using flexboxes has made it easy to create layouts that match a wide variety of screen sizes. However, she is concerned that under the single column layout used for mobile devices the daily schedule appears first before any description of the classes. She would like the daily schedule to appear at the bottom of the page. She asks if you can modify the layout to achieve this.

Reordering Page Content with Flexboxes

One of the principles of web page design is to, as much as possible, separate the page content from page design. However, a basic feature of any design is the order in which the content is displayed. Short of editing the content of the HTML file, there is not an easy way to change that order.

That at least was true before flexboxes. Under the flexbox model you can place the flex items in any order you choose using the following `order` property

```
order: value;
```

where `value` is an integer where items with smaller `order` values are placed before items with larger `order` values. For example, the following style arranges the `div` elements starting first with `item2`, followed by `item3`, and ending with `item1`. This is true regardless of how those `div` elements have been placed in the HTML document.

```
div#item1 {order: 100;}  
div#item2 {order: -1;}  
div#item3 {order: 5;}
```

TIP

If flex items have the same order value, they are arranged in document order.

Note that order values can be negative. The default order value is 0.

For complete cross-browser support, you can apply the following browser extensions with flex item ordering:

```
-webkit-box-ordinal-group: value;  
-moz-box-ordinal-group: value;  
-ms-flex-order: value;  
-webkit-order: value;  
order: value;
```

Most current browsers support the CSS specifications, so you will limit your code to those properties.

REFERENCE

Reordering a Flex Item

- To reorder a flex item, apply the style

```
order: value;
```

where `value` is an integer where items with smaller `order` values are placed before items with larger `order` values.

For mobile devices, Marjorie wants the page header displayed first, followed by the main section, the `aside` element, and ending with the page footer. Add style rules now to the mobile device media query in the `tf_flex.css` style sheet to reorder the flex items.

To lay out the topic articles and navigation list:

- 1. Return to the **tf_flex.css** file in your editor and go to the Mobile Devices media query.
- 2. Add the following style rules, indented to offset them from the braces in the media query:

```
aside {
  order: 99;
}
footer {
  order: 100;
}
```

Note that the other flex items will have a default order value of 0 and thus will be displayed in document order before the `aside` and `footer` elements.

Figure 5–37 highlights the style rules to set the order of the `aside` and `footer` elements.

Figure 5–37

Setting the order of a flex item

```
/*
=====
Mobile Styles: 0 to 480px
=====
*/
@media only screen and (max-width: 480px) {
  aside {
    order: 99;
  }
  footer {
    order: 100;
  }
}
```

places the aside element before the body footer

places the body footer at the end of the flexbox

- 3. Save your changes to the file and then reload the `tf_prek.html` file in your web browser.
- 4. Reduce the width of the browser window below 480 pixels to show the mobile layout. Verify that the class schedule now appears at the bottom of the file directly before the body footer.

You've completed the ordering and flex layout of the Pre-K Classes page. You'll conclude your review of flexboxes by examining how flex items can be arranged within the flexbox container.

Exploring Flexbox Layouts

You can control how flex items are laid out using the `justify-content`, `align-items`, and `align-content` properties. You examine each property to see how flexboxes can be used to solve layout problems that have plagued web designers for many years.

Aligning Items along the Main Axis

Recall from Figure 5–26 that flexboxes have two axes: the main axis along which the flex items flow and the cross axis, which is perpendicular to the main axis. By default, flex items are laid down at the start of the main axis. To specify a different placement, apply the following `justify-content` property

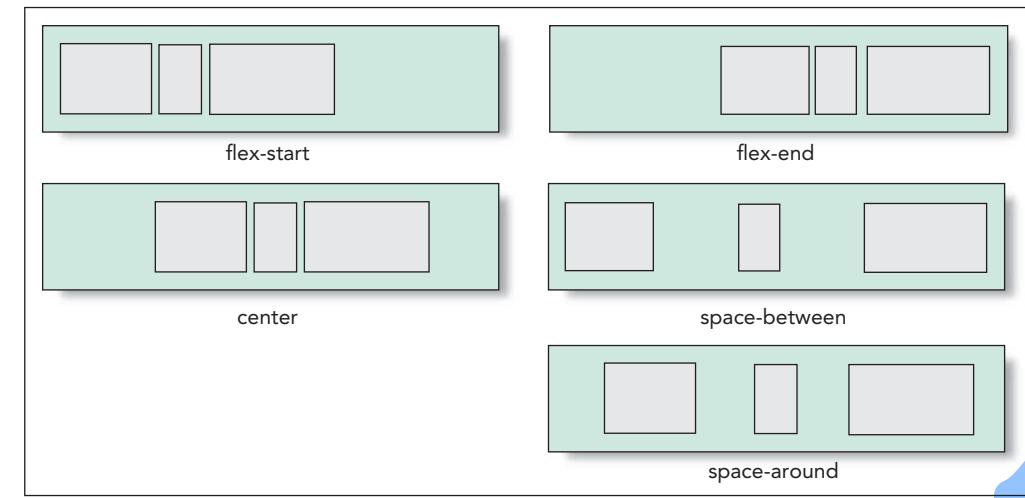
```
justify-content: placement;
```

where `placement` is one of the following keywords:

- `flex-start` Items are positioned at the start of the main axis (the default).
- `flex-end` Items are positioned at the end of the main axis.
- `center` Items are centered along the main axis.
- `space-between` Items are distributed evenly with the first and last items aligned with the start and end of the main axis.
- `space-around` Items are distributed evenly along the main axis with equal space between them and the ends of the flexbox.

Figure 5–38 shows the impact of different `justify-content` values on a flexbox oriented horizontally.

Figure 5–38 Values of the `justify-content` property



Remember that, because items can flow in any direction within a flexbox, these diagrams will look different for flexboxes under column orientation or when the content flows from the right to the left. Note that the `justify-content` property has no impact when the items are flexed to fill the entire space. It is only impactful for flex items with fixed sizes that do not fill in the entire flexbox.

Aligning Flex Lines

The `align-content` property is similar to the `justify-content` property except that it arranges multiple lines of content along the flexbox's cross axis. The syntax of the `align-content` property is:

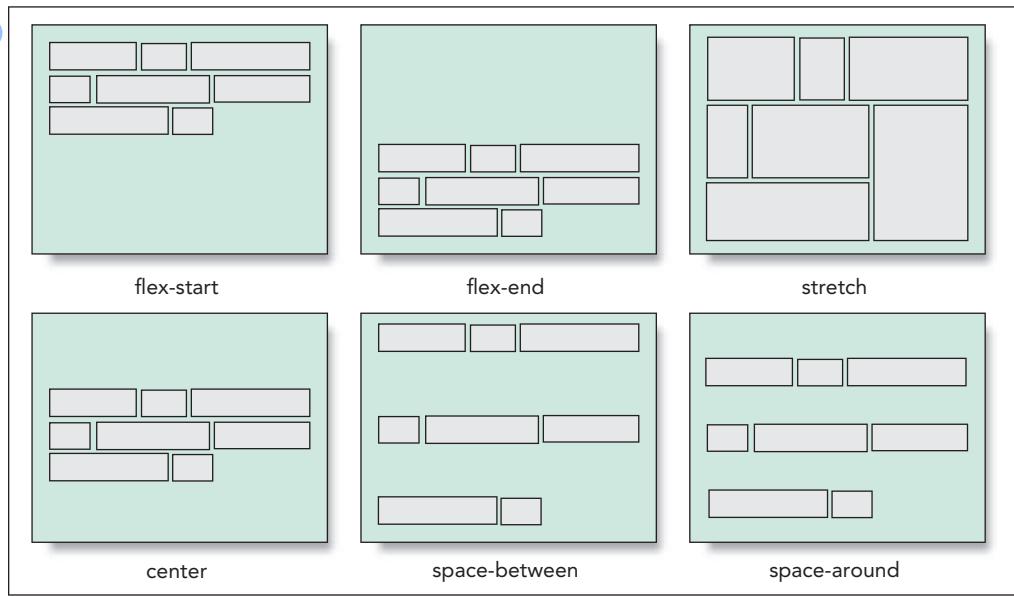
```
align-content: value;
```

where `value` is one of the following keywords:

- `flex-start` Lines are positioned at the start of the cross axis.
- `flex-end` Lines are positioned at the end of the cross axis.
- `stretch` Lines are stretched to fill up the cross axis (the default).
- `center` Lines are centered along the cross axis.
- `space-between` Lines are distributed evenly with the first and last lines aligned with the start and end of the cross axis.
- `space-around` Lines are distributed evenly along the cross axis with equal space between them and the ends of the cross axis.

Figure 5–39 displays the effect of the `align-content` values on three lines of flex items arranged within a flexbox.

Figure 5–39 `Values of the align-content property`



Note that the `align-content` property only has an impact when there is more than one line of flex items, such as occurs when wrapping is used with the flexbox.

Aligning Items along the Cross Axis

Finally, the `align-items` property aligns each flex item about the cross axis, having the syntax

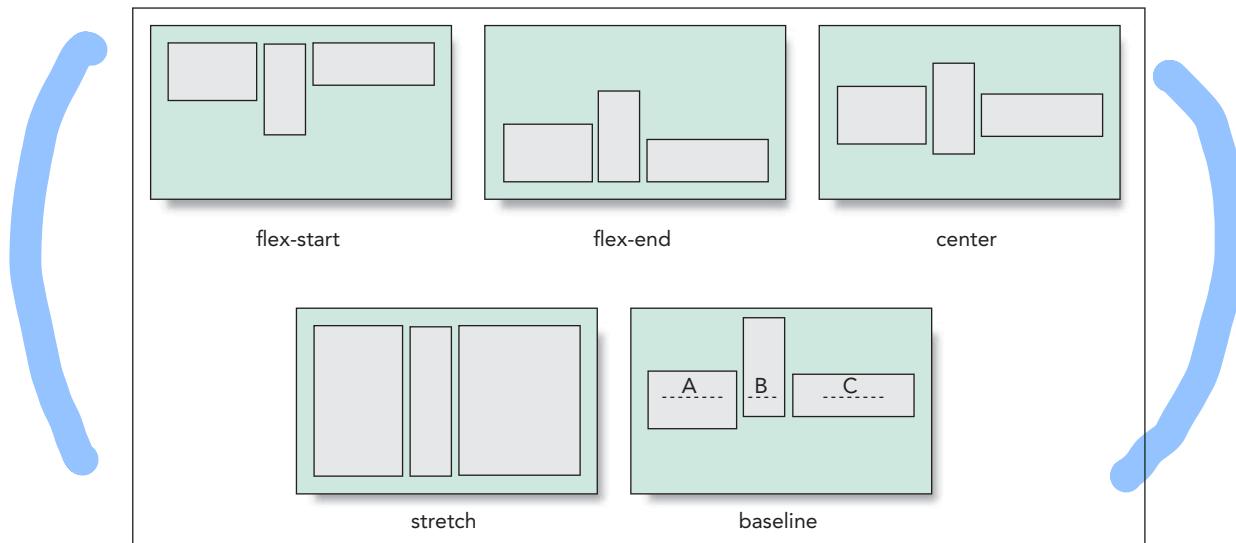
```
align-items: value;
```

where `value` is one of the following keywords:

- `flex-start` Items are positioned at the start of the cross axis.
- `flex-end` Items are positioned at the end of the cross axis.
- `center` Items are centered along the cross axis.
- `stretch` Items are stretched to fill up the cross axis (the default).
- `baseline` Items are positioned so that the baselines of their content align.

Figure 5–40 displays the effect of the `align-items` values on three flex items placed within a single line.

Figure 5–40 **Values of the align-items property**



Note that the `align-items` property is only impactful when there is a single line of flex items. With multiple lines, you use the `align-content` property to layout the flexbox content. To align a single item out of a line of flex items, use the following `align-self` property

```
align-self: value;
```

where `value` is one of the alignment choices supported by the `align-items` property. For example, the following style rule places the footer at the end of the flexbox cross axis, regardless of the placement of the other flex items.

```
footer {
    align-self: flex-end;
}
```

Both the `align-content` and `align-items` properties have a default value of `stretch` so that the flex items are stretched to fill the space along the cross-axis. The effect is that all flex items within a row will share a common height. This can be observed earlier in Figure 5–36 in which all of the article boxes have the same height, regardless of their content. It's difficult to achieve this simple effect in a grid layout unless the height of each item is explicitly defined, but flexboxes do it automatically.

INSIGHT

Solving the Centering Problem with Flexboxes

One of the difficult layout challenges in web design is vertically centering an element within its container. While there are many different fixes and “hacks” to create vertical centering, it has not been easily achieved until flexboxes. By using the `justify-content` and `align-items` properties, you can center an object or group of objects within a flexbox container. For example, the following style rule centers the child elements of the `div` element both horizontally and vertically:

```
div {  
    display: flex;  
    justify-content: center;  
    align-content: center;  
}
```

For a single object or a group of items on a single line within a container, use the `align-items` property as follows:

```
div {  
    display: flex;  
    justify-content: center;  
    align-items: center;  
}
```

You can also use the `align-self` property to center one of the items in the flexbox, leaving the other items to be placed where you wish.

Creating a Navicon Menu

A common technique for mobile websites is to hide navigation menus but to indicate their presence with a **navicon**, which is a symbol usually represented as three horizontal lines . When the user hovers or touches the icon, the navigation menu is revealed.

Marjorie has supplied you with a navicon image that she wants you to use with the mobile layout of the Pre-K Classes page. Add this image to the Pre-K Classes web page within the navigation list in the body header.

To insert the navicon image:

- 1. Return to the `tf_prek.html` file in your editor.
- 2. Directly after the opening `<nav>` tag in the body header, insert the following hypertext link and inline image.

```
<a id="navicon" href="#">  
      
</a>
```

Figure 5–41 highlights the code to create the navicon.

Figure 5–41

Inserting the navicon

```

<nav class="horizontal">
  <a id="navicon" href="#"></a>
  <ul>
    <li><a href="tf_home.html">Home</a></li>
    <li><a href="#">Infants</a></li>
    <li><a href="#">Toddlers</a></li>
    <li><a href="#" id="currentPage">Pre-K</a></li>
    <li><a href="#">After School</a></li>
  </ul>
</nav>

```

navicon image

Next, you'll insert the styles to hide and display the contents of the navigation list in a style sheet named `tf_navicon.css`. You'll apply the same styles for navicon that you used in the last session to hide and display the navigation submenus in the Trusted Friends home page. As with those menus, you'll use the `hover` pseudo-class to display the navigation list links whenever the user hovers over the navicon, or in the case of mobile devices, touches the navicon. Add these styles now.

To add styles for the navicon image:

- 1. Within the document head of the `tf_prek.html` file, add a link to the **`tf_navicon.css`** style sheet file after the link for the `tf_flex.css` file. Save your changes to the file.
- 2. Use your editor to open the **`tf_navicon_txt.css`** files from the `html05` ▶ tutorial folder. Enter **`your name`** and **`the date`** in the comment section of the file and save it as **`tf_navicon.css`**.
- 3. By default, the navicon will be hidden from the user. Go to the Base Styles section and add the following style rule:

```

a#navicon {
  display: none;
}

```

- 4. The navicon will be displayed only for mobile devices. Go to the media query for mobile devices and add the following style rule to display the navicon.

```

a#navicon {
  display: block;
}

```

- 5. When the navicon is displayed, you want the contents of the navigation list to be hidden. Add the following style rule within the mobile device media query:

```

nav.horizontal ul {
  display: none;
}

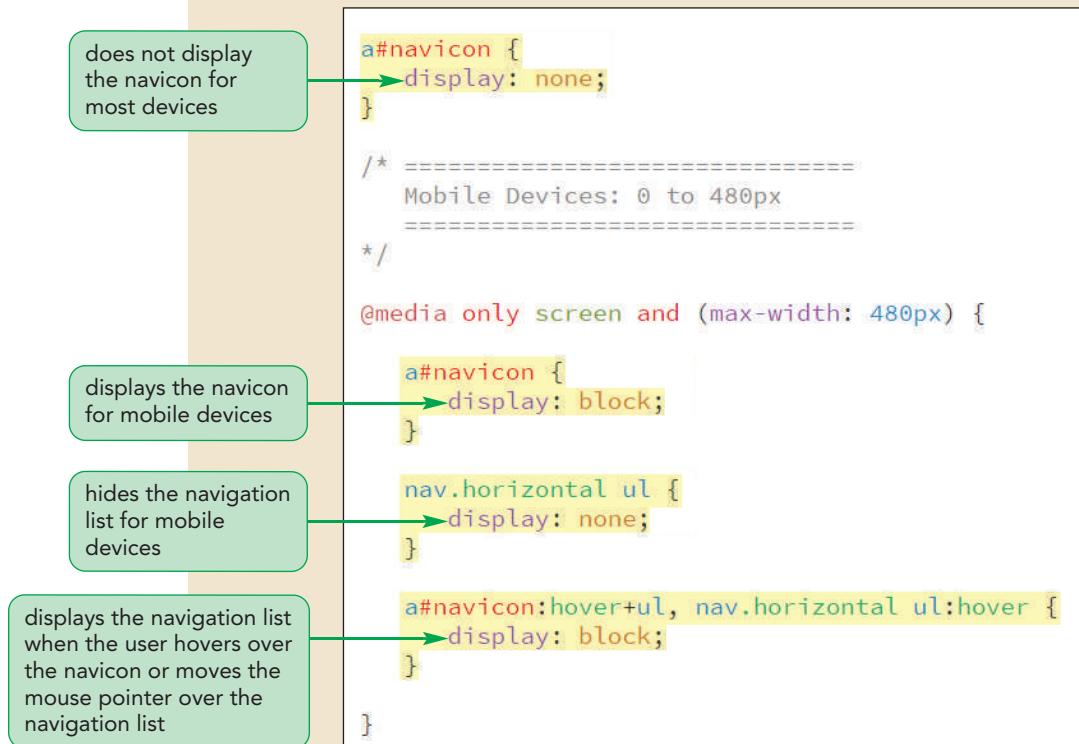
```

- 6. Finally, add the following style rule to the mobile device query that displays the contents of the navigation list when the user hovers over the navicon or the contents of the navigation list.

```
a#navicon:hover+ul, nav.horizontal ul:hover {
  display: block;
}
```

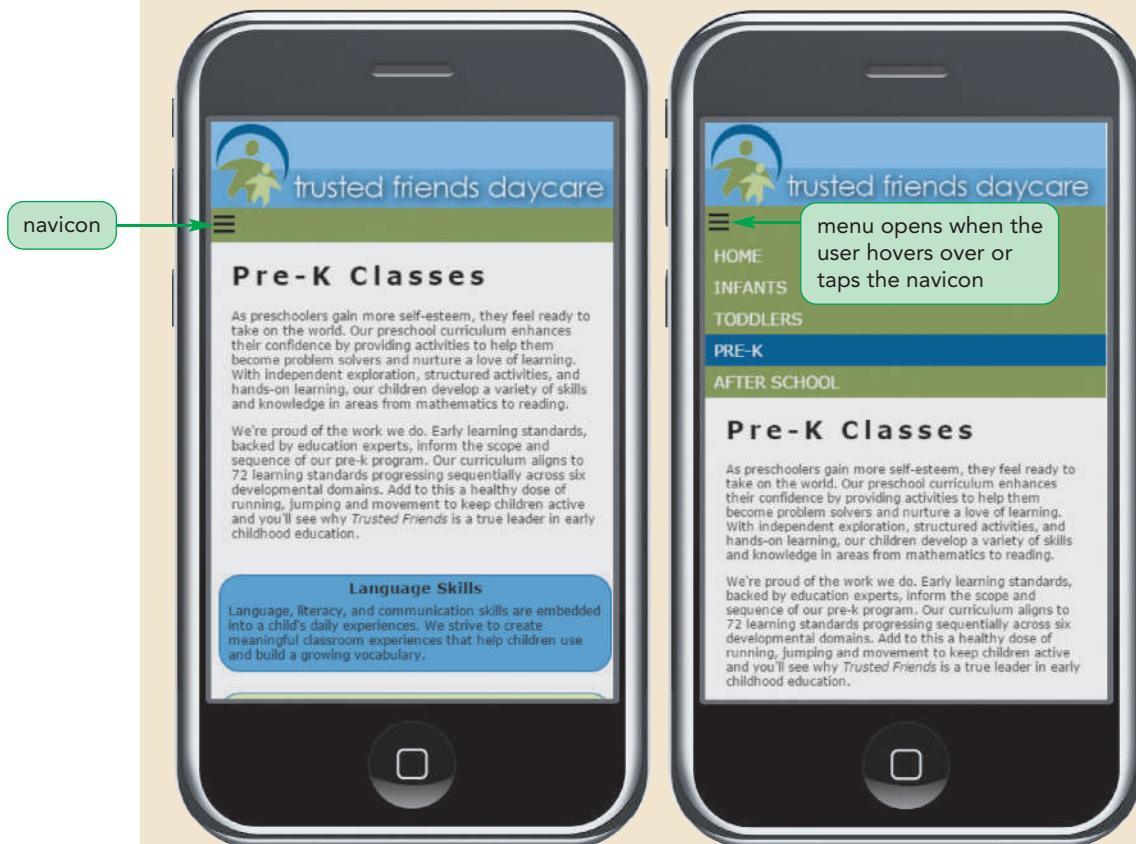
Figure 5–42 highlights the style rules for the navicon hypertext link.

Figure 5–42 Style rules for the navicon image



- 7. Save your changes to the file and then reload the tf_prek.html file in your browser or mobile devices. Resize the viewport as needed to display the mobile layout.
- 8. Verify that as you hover over or touch the navicon, the navigation list appears, as shown in Figure 5–43.

Figure 5–43 Action of the navicon for mobile devices



BenBois/openclipart

- 9. Verify that hovering over or touching other parts of the page hides the navigation list.

The methods you used in this tutorial to create pulldown menus and navicon menus represent what you can accomplish when limited to CSS and the hover pseudo-class. As you increase your skill and knowledge of HTML, you'll learn other, more efficient ways of creating mobile navigation menus using program scripts and web frameworks. If you want to explore how to take advantage of these tools, search the web for navicon libraries of prewritten code that can be inserted into your website.



PROSKILLS

Written Communication: Speeding Up Your Website by Minifying and Compressing

Once your website is working and you are ready to distribute it to the web, you have one task remaining: minifying your code. **Minifying** refers to the process of removing unnecessary characters that are not required for your site to execute properly. For example, the following text in a CSS file contains comments and line returns and blank spaces, which makes the text easy to read, but these features are not required and have no impact on how the browser renders the page:

```
/* Tablet Styles */

nav.horizontal > ul > li {
  display: block;
}
```

A minified version of this code removes the comment and the extraneous white-space characters leaving the following compact version:

```
nav.horizontal>ul>li{display:block;}
```

Minifying has several important advantages:

- Minifying reduces the amount of bandwidth required to retrieve the website because the files are smaller.
- The smaller minified files load faster and are faster to process because extraneous code does not need to be parsed by the browser.
- A faster site provides a better user experience.
- Smaller files means less server space required to host the website.
- Search engines, such as Google, evaluate your website based on page load speed and will downgrade sites with bloated code that take too long to load.

There are several free tools available on the web to automate the minification process including CSS Minifier, Compress HTML, HTML Minifier, and CSS Compressor. Also, many HTML editors include built-in minifying tools. Remember, a minified file is still a text file and can be read (though with difficulty) in a text editor.

To further reduce your file sizes, consider compressing your files using utilities like Gzip. A compressed file is no longer in text format and must be uncompressed before it is readable. All modern browsers support Gzip compression for files retrieved from a server. Make sure you know how to properly configure your web server to serve Gzip-compressed file in a readable format to the browser.

The process of minifying your files is irreversible, so make sure you retain the version with the text in a readable format and all of your comments preserved. Most minifying and compression tools will make a backup of your original files.

You've completed your work on the design of the Pre-K Classes page for Trusted Friends Daycare. In the next session, you'll explore other uses of media queries by designing a page for printed output. You may close your files now.

Session 5.2 Quick Check

1. Which of the following is *not* a style to display an element as a flexbox?
 - a. `display: -chrome-flex;`
 - b. `display: -webkit-flex;`
 - c. `display: -webkit-box`
 - d. `display: -ms-flexbox;`
2. To display items within a flexbox in a column filled from the bottom upward, use:
 - a. `flex-direction: column up;`
 - b. `flex-direction: column-bottom;`
 - c. `flex-direction: column-reverse;`
 - d. `flex-direction: column-to-top;`
3. To set the initial size of a flexbox item to 250 pixels, use:
 - a. `flex-size: 250px;`
 - b. `flex-basis: 250px;`
 - c. `flex: 250px;`
 - d. `flex-from: 250px;`
4. To set the growth rate of a flexbox item to a rate of 4, use:
 - a. `flex-rate: 4;`
 - b. `flex: 4x;`
 - c. `flex-growth: 4;`
 - d. `flex-grow: 4;`
5. Which of the following sets the `div` element to be equal in size regardless of the size of the flexbox container?
 - a. `div {flex: equal;}`
 - b. `div {flex: 1 1 100px;}`
 - c. `div {flex: 1 1 0px;}`
 - d. `div {flex: 0 0 0px;}`
6. To reorder the placement of a flex item within its flexbox, use:
 - a. `flex-reorder`
 - b. `flex-move`
 - c. `flex-basis;`
 - d. `order`
7. To center flex items along the flexbox's main axis, use:
 - a. `justify-content: center;`
 - b. `align-content: center;`
 - c. `flex-position: center;`
 - d. `flex-main: center;`
8. To center flex items along the flexbox's cross axis, use:
 - a. `justify-content: center;`
 - b. `align-content: center;`
 - c. `flex-position: center;`
 - d. `flex-main: center;`

Session 5.3 Visual Overview:

```
nav.horizontal, aside, footer {
  display: none;
}
```

The `display: none;` property is set to none for objects you don't want printed.

```
@page {
  size: 8.5in 11in portrait;
  margin: 0.5in;
}
```

The `@page` rule defines the size and margins of the printed page.

```
h1 {
  font-size: 28pt;
  line-height: 30pt;
  margin: 0.3in 0in 0.2in;
}
```

For print layouts, fonts should be sized in points and widths and heights expressed in inches or centimeters.

```
a::after {
  content: " (" attr(href) ")";
  font-weight: bold;
  word-wrap: break-word;
}
```

Use the `after` pseudo-element along with the `content` property to display the text of all hypertext URLs.

```
article:nth-of-type(n+2) {
  page-break-before: always;
}
```

Use the `page-break-before: always;` property to insert page breaks before elements.

```
img, ol, ul {
  page-break-inside: avoid;
}
```

Use the `page-break-inside: avoid;` property to prohibit page breaks within an element.

```
p {
  orphans: 3;
  widows: 3;
}
```

Use the `orphans: 3;` and `widows: 3;` properties to limit the number of lines stranded at the top and bottom of a page.

Print Styles

Page size is set at 8.5 inches by 11 inches with a 0.5 inch margin in portrait orientation.

Trusted Friends: Article of Interest



trusted friends daycare

An Accredited Center



Pressmaster/
Shutterstock.com

At Trusted Friends we believe that every child is capable of excellence. That is why we are committed to pursuing and maintaining our status as an accredited daycare center. By seeking national accreditation, you know that Trusted Friends is striving to give your family the very best daycare experience.

What is Accreditation?

Every daycare center must meet the state's **minimum** license requirements. We go beyond that. When a daycare center is awarded national accreditation they are meeting a higher standard that demonstrates its expertise in:

- Classroom Management
- Curriculum Development
- Health and Safety
- Parent Support
- Community Involvement
- Teacher Certification
- Administrative Oversight
- Financial Statements

page 1

Trusted Friends: Article of Interest

Our commitment to accreditation gives you assurance we provide a positive educational experience for your child.

How does Accreditation Work?

Every other year we go through an intense review by recognized and esteemed national accreditation agencies. Their positive reports (available for inspection) confirm that we are providing a clean, safe, and positive environment for our children. Accreditation verifies that our teachers are qualified and fully engaged in giving our children a first-class educational experience.

Once we've completed the entire accreditation self-study process, trained professionals from our accrediting agencies conduct on-site visits to validate our compliance with national early childhood education standards. But accreditation doesn't just take place every two years. It's an ongoing process of self-evaluation, discussion, and parental reviews.

We encourage parents to help us improve our center and become better stewards for their children. You can part of the accreditation process as we work together to make Trusted Friends a great neighborhood center.

Who Provides Accreditation?

There are several national organizations that provide accreditation services. Who a center chooses for oversight is important. Trusted Friends pursues national accreditation from three of the most respected national early childhood accreditation agencies:

- National Association for Youth Care (<http://www.example.com/nayc>)
- United Accreditation for Early Care and Education (<http://www.example.com/naece>)
- National Daycare Accreditation (<http://www.example.com/nda>)

Feel free to contact us to discuss accreditation and learn more about our standards for care and education.

page 2

Hypertext URLs are displayed in bold after the hypertext link.

Trusted Friends: Article of Interest

Our Community



Gladskikh Tatiana/
Shutterstock.com

Trusted Friends is committed to improving the lives of children in our community. Our expertise in caring for the children at our daycare center gives us a unique understanding of child development, education issues, and parenting. Trusted Friends has partnered with several community organizations that advocate for poor and needy children and families.

We don't think of it as charity. It's part of our calling.

Improving Literacy

Part of Trusted Friends' mission is to promote literacy, which is key to education and a fulfilling life. We support reading programs and national literacy efforts initiated at both the local and national level. These efforts include providing early access to books and other reading material. We are also in the Raised by Reading (<http://www.example.com/rbr>) program, helping parents share the reading experience with their children.

Promoting Partnerships

We are proud of our support for the Big Siblings (<http://www.example.com/bigs>) organization. Several of our educators are Big Sibling mentors and we provide meeting space and monthly activities for this fine group. We are also deeply involved with the Young Care Nursery (<http://www.example.com/ycn>) organization, working to prevent child abuse and neglect. We partner with other caregivers committed to strengthening families in the community. For example we are a charter member of Sunflower Friends (<http://www.example.com/sf>), which creates learning and enrichment opportunities for underprivileged children, helping them to realize their potential and recognize their inherent dignity.

Please contact us if you believe that Trusted Friends can be a partner with your group in improving the lives of children and families in our community.

page 3

Page break is inserted before the article element, starting it on a new page.

Designing for Printed Media

So far your media queries have been limited to screens of different widths. In this session you'll explore how to apply media queries to print devices and work with several CSS styles that apply to printed output. To do this you'll create a **print style sheet** that formats the printed version of your web document.

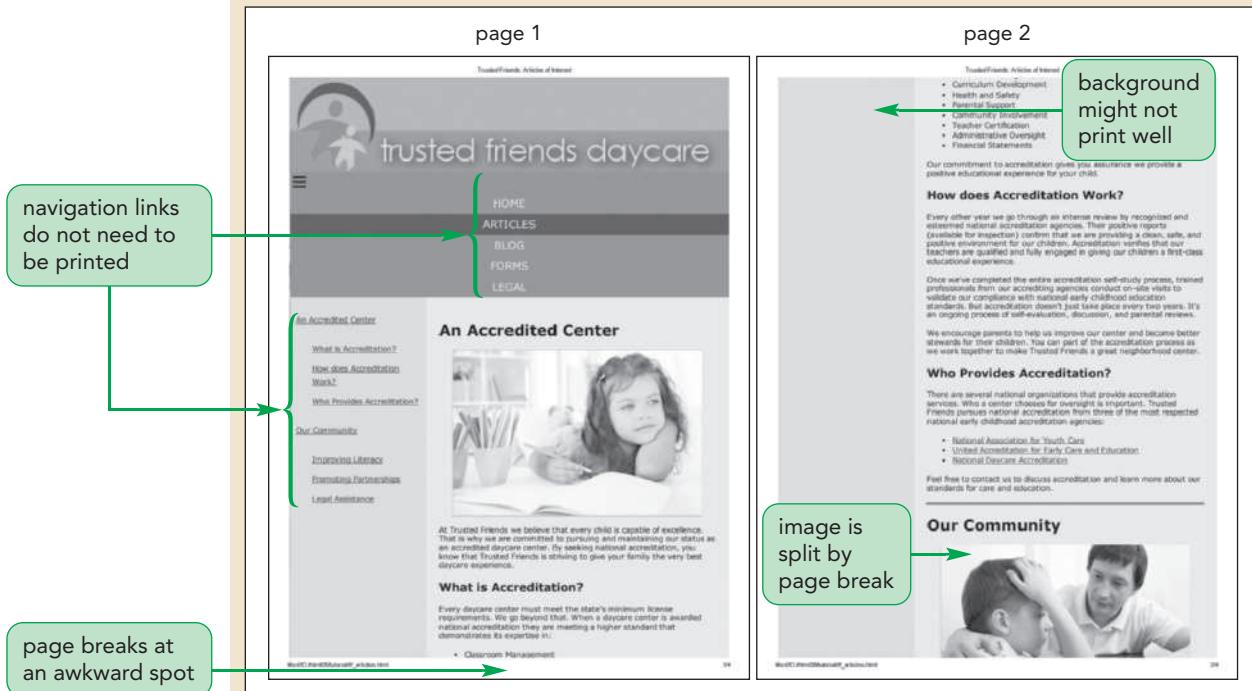
Previewing the Print Version

Marjorie has created a page containing articles of interest for parents at Trusted Friends Daycare. She has already written the page content and the style sheets for mobile, tablet, and desktop devices. Open the articles document now.

To open the Articles of Interest page:

- 1. Use your editor to open the **tf_articles_txt.html** file from the **html05** tutorial folder. Enter **your name** and **the date** in the comment section of the file and save it as **tf_articles.html**.
 - 2. Within the document head, create links to the **tf_reset.css** and **tf_styles3.css** style sheet files in that order.
 - 3. Scroll through the document to become familiar with its contents and then save your changes to file, but do not close it.
 - 4. Open the **tf_articles.html** file in your web browser.
 - 5. Take some time to view the contents of the page under different screen resolutions, noting how Marjorie has used responsive design to create different page layouts based on the screen width.
- Now, you'll examine how Marjorie's page will appear when printed.
- 6. Use the Print Preview command within your browser to preview how this page will appear when printed. Figure 5–44 shows a preview of the first two pages of the print version using a black and white printer.

Figure 5–44 Print version of the Articles of Interest page



Trouble? Depending on your browser and printer, your print preview might appear different from the preview shown in Figure 5–44.

Browsers support their own internal style sheet to format the print versions of the web pages they encounter. However, their default styles might not always result in the best printouts. Marjorie points out that the print version of her page has several significant problems:

- The printed version includes two navigation lists, neither of which have a purpose in a printout.
- Page breaks have been placed in awkward places, splitting paragraphs and images in two.
- Background colors, while looking good on a screen, might not print well.

Marjorie would like you to design a custom print style sheet that fixes these problems by removing unnecessary page elements and choosing page breaks more intelligently.

Applying a Media Query for Printed Output

To apply a print style sheet, you use the `media` attribute in your `link` elements to target style sheets to either screen devices or print devices. Modify the `tf_articles.html` file now to access a new style sheet named `tf_print.css` into which you include your print styles.

To access a print style sheet:

- 1. Use your editor to open the `tf_print_txt.css` file from the `html05 > tutorial` folder. Enter **your name** and **the date** in the comment section and save it as `tf_print.css`.
- 2. Return to the `tf_articles.html` file in your editor. Add the attribute `media="all"` to the `link` element for the `tf_reset.css` style sheet to apply it to all devices.
- 3. Add the attribute `media="screen"` to the `link` element for the `tf_styles3.css` style sheet to apply it only to screen devices.
- 4. Add the following `link` element for print styles:

```
<link href="tf_print.css" rel="stylesheet" media="print" />
```

Figure 5–45 highlights the revised `link` elements in the file.

Figure 5–45

Style sheets for different devices

```
<title>Trusted Friends: Articles of Interest</title>
<meta charset="utf-8" />
<meta name="viewport" content="width=device-width, initial-scale=1" />
<link href="tf_reset.css" rel="stylesheet" media="all" />
<link href="tf_styles3.css" rel="stylesheet" media="screen" />
<link href="tf_print.css" rel="stylesheet" media="print" />
</head>
```

styles for all devices

styles for print devices

styles for screen devices

- 5. Save your changes to the file and close it.

You'll start designing the print version of this page by hiding those page elements that should not be printed, including the navigation list, the `aside` element, and the body footer.

To hide elements in the print version:

- 1. Return to the `tf_print.css` file in your editor.
- 2. Go to the Hidden Objects section and add the following style rule:

```
nav.horizontal, aside, footer {  
    display: none;  
}
```

Figure 5–46 highlights the style rule to hide page elements.

Figure 5–46 **Hiding page elements for printing**

sets the display of the navigation list, `aside` element, and body footer to do not display

```
/* Hidden Objects */  
nav.horizontal, aside, footer {  
    display: none;  
}
```

- 3. Save your changes to the file and then reload the `tf_articles.html` file in your browser and preview the printed output. Verify that the navigation lists, `aside` elements, and body footer are not displayed in the printed version.

Next, you'll define the page size of the print version of this document.

Working with the `@page` Rule

In CSS every printed page is defined as a **page box**, composed of two areas: the **page area**, which contains the content of the document, and the **margin area**, which contains the space between the printed content and the edges of the page.

Styles are applied to the page box using the following `@page` rule

```
@page {  
    style rules  
}
```

where `style rules` are the styles applied to the page. The styles are limited to defining the page size and the page margin. For example, the following `@page` rule sets the size of the page margin to 0.5 inches:

```
@page {  
    margin: 0.5in;  
}
```

The page box does not support all of the measurement units you've used with the other elements. For example, pages do not support the `em` or `ex` measurement units. In general, you should use measurement units that are appropriate to the dimensions of your page, such as inches or centimeters.

Copyright 2021 Cengage Learning. All Rights Reserved. May not be copied, scanned, or duplicated, in whole or in part. Due to electronic rights, some third party content may be suppressed from the eBook and/or eChapter(s). Editorial review has deemed that any suppressed content does not materially affect the overall learning experience. Cengage Learning reserves the right to remove additional content at any time if subsequent rights restrictions require it.

Setting the Page Size

Because printed media can vary in size and orientation, the following `size` property allows web authors to define the dimensions of the printed page

```
size: width height;
```

TIP

Users can override the page sizes and orientations set in `@page` rule by changing the options in their print dialog box.

where `width` and `height` are the width and height of the page. Thus to define a page that is 8.5 inches wide by 11 inches tall with a 1-inch margin, you would apply the following style rule:

```
@page {
    size: 8.5in 11in;
    margin: 1in;
}
```

You can replace the `width` and `height` values with the keyword `auto` (to let browsers determine the page dimensions) or `inherit` (to inherit the page size from the parent element). If a page does not fit into the dimensions specified in the `@page` rule, browsers will either rotate the page or rescale it to fit within the defined page size.

Using the Page Pseudo-Classes

By default, the `@page` rule is applied to every page of the printed output. However, if the output covers several pages, you can define different styles for different pages by adding the following pseudo-class to the `@page` rule:

```
@page:pseudo-class {
    style rules
}
```

where `pseudo-class` is `first` for the first page of the printout, `left` for the pages that appear on the left in double-sided printouts, or `right` for pages that appear on the right in double-sided printouts. For example, if you are printing on both sides of the paper, you might want to create mirror images of the margins for the left and right pages of the printout. The following styles result in pages in which the inner margin is set to 5 centimeters and the outer margin is set to 2 centimeters:

```
@page:left {margin: 3cm 5cm 3cm 2cm;}
@page:right {margin: 3cm 2cm 3cm 5cm;}
```

Page Names and the Page Property

To define styles for pages other than the first, left, or right, you first must create a page name for those styles as follows

```
@page name {
    style rules
}
```

where `name` is the label given to the page. The following code defines a page style named `wideMargins` used for pages in which the page margin is set at 10 centimeters on every side:

```
@page wideMargins {
    margin: 10cm;
}
```

Once you define a page name, you can apply it to any element in your document. The content of the element will appear on its own page, with the browser automatically

inserting page breaks before and after the element if required. To assign a page name to an element, you use the following `page` property

```
selector {  
    page: name;  
}
```

where `selector` identifies the element that will be displayed on its own page, and `name` is the name of a previously defined page style. Thus the following style rule causes all block quotes to be displayed on separate page(s) using the styles previously defined as the `wideMargins` page:

```
blockquote {  
    page: wideMargins;  
}
```

REFERENCE

Creating and Applying Page Styles

- To define a page box for the printed version of a document, use the CSS rule

```
@page {  
    size: width height;  
}
```

where `width` and `height` are the width and height of the page.

- To define the page styles for different output pages, use the rule

```
@page:pseudo-class {  
    style rules  
}
```

where `pseudo-class` is `first` for the first page of the printout, `left` for the pages that appear on the left in double-sided printouts, or `right` for pages that appear on the right in double-sided printouts.

- To create a named page for specific page styles, apply the rule

```
@page name {  
    style rules  
}
```

where `name` is the label assigned to the page style.

- To apply a named page style, use the rule

```
selector {  
    page: name;  
}
```

where `selector` identifies the element that will be displayed on its own page, and `name` is the name of a previously defined page style.

You'll use the `@page` rule to define the page size for the printed version of the Articles of Interest document. Marjorie suggests that you set the page size to 8.5×11 inches with 0.5-inch margins.

To define the printed page size:

- 1. Return to the `tf_print.css` file in your editor.
- 2. Go to the Page Box Styles section and add the following rule:

```
@page {
    size: 8.5in 11in;
    margin: 0.5in;
}
```

Figure 5–47 highlights the rule to set the page size.

Figure 5–47

Setting the page size

sets the page to 8.5 inches wide by 11 inches long

sets the margin to 0.5 inches around the page content

- 3. Save your changes to the file.

With printed output, widths and heights are measured not in pixels but in inches or centimeters. Font sizes are not measured in pixels but rather in points. With that in mind, create styles to format the sizes of the text and graphics on the page.

To format the printed text:

- 1. Go to the Typography Styles section and insert the following styles to format the appearance of h1 and h2 headings and paragraphs:

```
h1 {
    font-size: 28pt;
    line-height: 30pt;
    margin: 0.3in 0in 0.2in;
}

h2 {
    font-size: 20pt;
    margin: 0.1in 0in 0.1in 0.3in;
}

p {
    font-size: 12pt;
    margin: 0.1in 0in 0.1in 0.3in;
}
```

- 2. Within the List Styles section, add the following style rules to format the appearance of unordered lists:

```
ul {
    list-style-type: disc;
    margin-left: 0.5in;
}
```

Figure 5–48 shows the typography and list styles in the print style sheet.

Figure 5–48

Typographical formats

```

/* Typography Styles */

h1 {
    font-size: 28pt;
    line-height: 30pt;
    margin: 0.3in 0in 0.2in;
}

h2 {
    font-size: 20pt;
    margin: 0.1in 0in 0.1in 0.3in;
}

p {
    font-size: 12pt;
    margin: 0.1in 0in 0.1in 0.3in;
}

/* List Styles */

ul {
    list-style-type: disc;
    margin-left: 0.5in;
}

```

font sizes are measured in points

format of h1 headings

format of h2 headings

format of paragraphs

format of unordered lists

margins are measured in inches

Next, you'll format the appearance of images on the page.

To format the printed images:

- 1. Within the Image Styles section, add the following style rule to format the appearance of inline images within each `article` element:

```

article img {
    border: 2px solid rgb(191, 191, 191);
    display: block;
    margin: 0.25in auto;
    width: 65%;
}

```

Figure 5–49 shows the style rule for inline images on the printed page.

Figure 5–49

Image formats

```

/* Image Styles */

article img {
    border: 2px solid rgb(191, 191, 191);
    display: block;
    margin: 0.25in auto;
    width: 65%;
}

```

displays all article images with a gray border, with a width of 65% of the page body, and centered horizontally

- 2. Save your changes to the style sheet and then reload the `tf_articles.html` file in your browser and preview the appearance of the printed page. Figure 5–50 shows the appearance of the first page printed using a black and white printer.

Figure 5–50

Preview of the first printed page

Trusted Friends: Articles of Interest



trusted friends daycare

An Accredited Center



print version of the images

At Trusted Friends we believe that every child is capable of excellence. That is why we are committed to pursuing and maintaining our status as an accredited daycare center. By seeking national accreditation, you know that Trusted Friends is striving to give your family the very best daycare experience.

What is Accreditation?

Every daycare center must meet the state's minimum license requirements. We go beyond that. When a daycare center is awarded national accreditation they are meeting a higher standard that demonstrates its expertise in:

- Classroom Management
- Curriculum Development
- Health and Safety
- Parental Support
- Community Involvement
- Teacher Certification
- Administrative Oversight
- Financial Statements

file:///D:/html05/tutorial/ff_articles.html

1/4

© Pressmaster/Shutterstock.com

Marjorie notices that all of the hyperlinks in the document appear in blue and underlined as determined by the default browser style. While this identifies the text as a hypertext link, it doesn't provide the reader any information about that link. She asks you to modify the style sheet to fix this problem.

Formatting Hypertext Links for Printing

Because printouts are not interactive, it's more useful for the reader to see the URL of a hypertext link so that he or she can access that URL at another time. To append the text of a link's URL to the linked text, you can apply the following style rule:

```
a::after {  
    content: " (" attr(href) ") ";  
}
```

TIP

Be sure to include blank spaces around the href value so that the URL does not run into the surrounding text.

This style rule uses the `after` pseudo-element along with the `content` property and the `attr()` function to retrieve the text of the `href` attribute and add it to the contents of the `a` element.

You should be careful when using this technique. Appending the text of a long and complicated URL will make your text difficult to read and might break your page layout if the text string extends beyond the boundaries of its container. One way to solve this problem is to apply the following `word-wrap` property to the URL text:

```
word-wrap: type;
```

where `type` is either `normal` (the default) or `break-word`. A value of `normal` breaks a text string only at common break points such as the white space between words. A value of `break-word` allows long text to be broken at arbitrary points, such as within a word, if that is necessary to make the text string fit within its container. Because a URL has no common break points such as blank spaces, applying the `break-word` option ensures that the text string of the URL will be kept to a manageable length by breaking it as needed to fit within the page layout.

REFERENCE

Formatting Hypertext for Printing

- To add the URL after a hypertext link, apply the style rule:

```
a::after {  
    content: " (" attr(href) ") ";  
}
```

- To automatically wrap the text of long URLs as needed, add the following style to the link text:

```
word-wrap: break-word;
```

Format the appearance of hypertext links in the document to display each link's URL and to display the hypertext links in a black bold font with no underlining, then use the `word-wrap` property to keep long URLs from extending beyond the boundaries of their container.

To format the hypertext links:

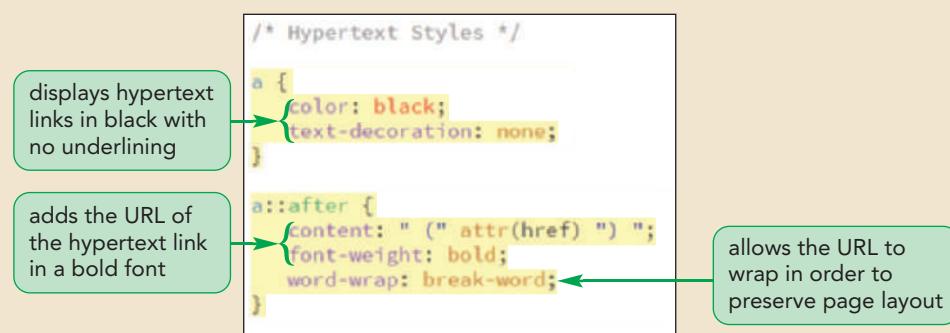
- 1. Return to the **tf_print.css** file in your editor and go to Hypertext Styles section, inserting the following styles to format the appearance of all hypertext links, appending the URL of each link:

```
a {
    color: black;
    text-decoration: none;
}

a::after {
    content: "(" attr(href) ")";
    font-weight: bold;
    word-wrap: break-word;
}
```

Figure 5–51 describes the style rules used to format printed hypertext links.

Figure 5–51 **Formatting printed hypertext links**



- 2. Save your changes to the style sheet and then reload the **tf_articles.html** file in your browser and preview the page printout. Figure 5–52 shows the appearance of the printed hypertext links found on the second page of Marjorie's printout.

Figure 5–52

Preview of the hypertext links on page 2

Trusted Friends: Articles of Interest

How does Accreditation Work?

Every other year we go through an intense review by recognized and esteemed national accreditation agencies. Their positive reports (available for inspection) confirm that we are providing a clean, safe, and positive environment for our children. Accreditation verifies that our teachers are qualified and full engaged in giving our children a first-class educational experience.

Once we've completed the entire accreditation self-study process, trained professionals from our accrediting agencies conduct on-site visits to validate our compliance with national early childhood education standards. But accreditation doesn't just take place every two years. It's an ongoing process of self-evaluation, discussion, and parental reviews. We encourage our parents to help us improve our center and become better stewards for their children.

Who Provides Accreditation?

Trusted Friends pursues national accreditation from three of the most respected national early childhood accreditation agencies:

- National Association for Youth Care (<http://www.example.com/nayc>)
- United Accreditation for Early Care and Education (<http://www.example.com/uaece>)
- National Daycare Accreditation (<http://www.example.com/nda>)

Feel free to contact us to discuss accreditation and learn more about our standards for care and education.

Our Community



Trusted Friends is committed to improving the lives of children in our community. Our expertise in caring for the children at our daycare center gives us a unique understanding of child development, education issues, and parenting. Trusted Friends has partnered with several community organizations that advocate for poor and needy children and families. We don't think of it as charity. It's part of our calling.

2/3

© Gladskikh Tatiana/Shutterstock.com

You can search the web for several free scripting tools that give you more options for how your URLs should be printed, including scripts that automatically append all URLs as footnotes at the end of the printed document.

Working with Page Breaks

When a document is sent to a printer, the browser determines the location of the page breaks unless that information is included as part of the print style sheet. To manually insert a page break either directly before or directly after an element, apply the following `page-break-before` or `page-break-after` properties:

```
page-break-before: type;
page-break-after: type;
```

where `type` has the following possible values:

- `always` Use to always place a page break before or after the element
- `avoid` Use to never place a page break
- `left` Use to place a page break where the next page will be a left page
- `right` Use to place a page break where the next page will be a right page
- `auto` Use to allow the printer to determine whether or not to insert a page break
- `inherit` Use to insert the page break style from the parent element

For example, if you want each `h1` heading to start on a new page, you would apply the following style rule to insert a page break before each heading:

```
h1 {
    page-break-before: always;
}
```

REFERENCE

Adding a Page Break

- To set the page break style directly before an element, apply the property
`page-break-before: type;`
 where `type` is `always`, `avoid`, `left`, `right`, `auto`, or `inherit`.
- To set the page break style directly after an element, apply
`page-break-after: type;`

After the first article, Marjorie wants each subsequent article to start on a new page. To select every article after the initial article, use the selector

```
article:nth-of-type(n+2)
```

which selects the second, third, fourth, and so on `article` elements in the document (see “Exploring the `nth-of-type` Pseudo-class” in Tutorial 2.) To ensure that each of the selected articles starts on a new page, insert the page break before the article using the following style rule:

```
article:nth-of-type(n+2) {
    page-break-before: always;
}
```

Add this style rule to the print style sheet now.

To print each article on a new page:

- 1. Go to the Page Break Styles section and insert the following style rule:

```
article:nth-of-type(n+2) {
    page-break-before: always;
}
```

Figure 5–53 highlights the style rule to insert the article page breaks.

Figure 5–53

Adding page breaks before the document articles

```
/* Page Break Styles */
article:nth-of-type(n+2) {
    page-break-before: always;
}
```

- 2. Save your changes to the file and then reload the `tf_articles.html` file in your browser and preview the printed page. Verify that the second article in the document on Community Involvement starts on a new page.

Next, you'll explore how to remove page breaks from the printed version of your web page.

INSIGHT

How Browsers Set Automatic Page Breaks

Browsers establish page breaks automatically, unless you manually specify the page breaks with a print style sheet. By default, browsers insert page breaks using the following guidelines:

- Insert all of the manual page breaks as indicated by the `page-break-before`, `page-break-after`, and `page-break-inside` properties
- Break the pages as few times as possible
- Make all pages that don't have a forced page break appear to have the same height
- Avoid page breaking inside page elements that have a border
- Avoid page breaking inside a web table
- Avoid page breaking inside a floating element

Other styles from the print style sheet are applied only after attempting to satisfy these constraints. Note that different browsers apply page breaks in different ways, so while you can apply general rules to your print layout, you cannot, at the current time, make the print versions completely consistent across browsers.

Preventing Page Breaks

You can prevent a page break by using the keyword `avoid` in the `page-break-after` or `page-break-before` properties. For example, the following style rule prevents page breaks from being added after any heading.

```
h1, h2, h3, h4, h5, h6 {
    page-break-after: avoid;
}
```

Unfortunately in actual practice, most current browsers don't reliably support prohibiting page breaks in this fashion. Thus, to prevent page breaks after an element, you will usually have to manually insert a page break before the element so that the element is moved to the top of the next page.

For other print layouts, you will want to prevent page breaks from being placed inside an element. This usually occurs when you have a long string of text that you don't want broken into two pages. You can prevent printers from inserting a page break by using the following `page-break-inside` property

```
page-break-inside: type;
```

where `type` is `auto`, `inherit`, or `avoid`. Thus, to prevent a page break from appearing within any image you can apply the following style rule:

```
img {  
    page-break-inside: avoid;  
}
```

Unlike the `page-break-before` and `page-break-after` properties, almost all current browsers support the use of the `avoid` keyword for internal page breaks.

REFERENCE

Preventing Page Breaks Inside an Element

- To prevent a page break from occurring within an element, apply the style:

```
page-break-inside: avoid;
```

Marjorie asks you to revise the print style sheet to prevent page breaks from occurring within images, ordered lists, and unordered lists.

To avoid page breaks:

- 1. Return to the `tf_print.css` file in your editor and go to the Page Break Styles section and insert the following style rule:

```
img, ol, ul {  
    page-break-inside: avoid;  
}
```

Figure 5–54 highlights the style rule to avoid page breaks in lists and images.

Figure 5–54

Avoiding line breaks within lists and images

```
/* Page Break Styles */  
  
article:nth-of-type(n+2) {  
    page-break-before: always;  
}  
  
img, ol, ul {  
    page-break-inside: avoid;  
}
```

avoids line breaks
within lists and images

- 2. Save your changes to the file.

Note that the `avoid` type does not guarantee that there will never be a page break within the element. If the content of an element exceeds the dimensions of the sheet of paper on which it's being printed, the browser will be forced to insert a page break.

Working with Widows and Orphans

Page breaks within block elements, such as paragraphs, can often leave behind widows and orphans. A widow is a fragment of text left dangling at the top of page, while an orphan is a text fragment left at the bottom of a page. Widows and orphans generally ruin the flow of the page text, making the document difficult to read. To control the size of widows and orphans, CSS supports the following properties:

```
widows: value;  
orphans: value;
```

where `value` is the number of lines that must appear within the element before a page break can be inserted by the printer. The default value is 2, which means that a widow or orphan must have at least two lines of text before it can be preceded or followed by a page break.

If you wanted to increase the size of widows and orphans to three lines for the paragraphs in a document, you could apply the style rule

```
p {  
    widows: 3;  
    orphans: 3;  
}
```

and the browser will not insert a page break if fewer than three lines of a paragraph would be stranded at either the top or the bottom of the page.

REFERENCE

Controlling the Size of Widows and Orphans

- To set the minimum size of widows (lines stranded at the top of a page), apply the property

```
widows: value;
```

where `value` is the number of lines that must appear at the top of the page before the page break.

- To set the minimum size of orphans (lines stranded at the bottom of a page), apply the property

```
orphans: value;
```

where `value` is the number of lines that must appear at the bottom of the page before the page break.

Use the `widows` and `orphans` properties now, setting their size to 3 for paragraphs in the printed version of the Articles of Interest page.

To avoid widows and orphans:

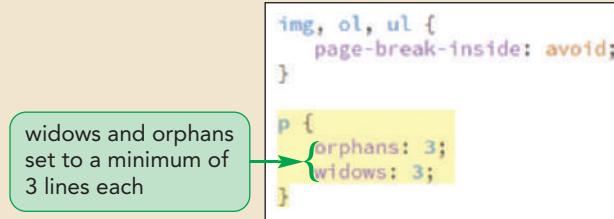
- 1. Within the Page Break Styles section of the **tf_print.css** file, add the following style rule.

```
p {
  orphans: 3;
  widows: 3;
}
```

Figure 5–55 highlights the style rule for setting the size of widows and orphans.

Figure 5–55

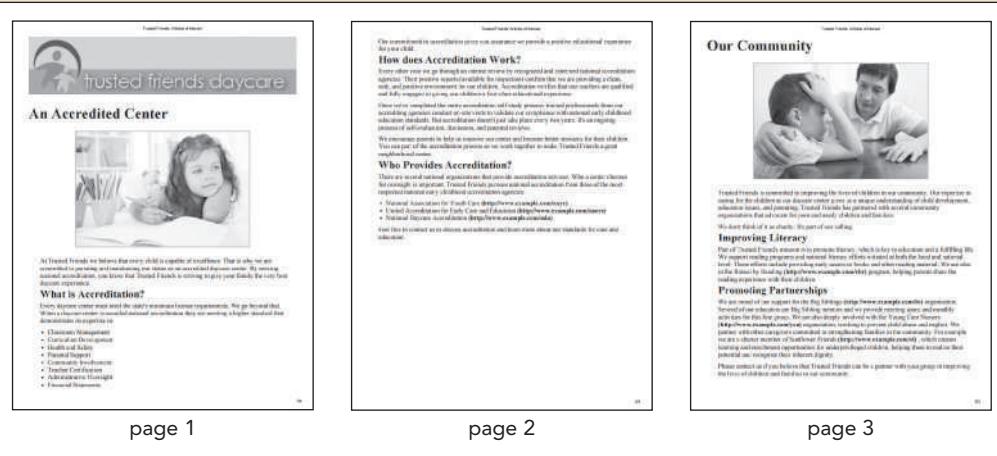
Setting the size of widows and orphans



- 2. Save your changes to the file and then reload the **tf_articles.html** file in your browser. Preview the appearance of the printed document. Figure 5–56 shows the final appearance of the printed version of this document.

Figure 5–56

Final print version of the document



© Pressmaster/Shutterstock.com; © Gladskikh Tatiana/Shutterstock.com;

Trouble? Depending on your browser and your default printer, your printed version may look slightly different from the one shown in Figure 5–56.

You've completed your work on the print styles for the Articles of Interest page. By modifying the default style sheet, you've created a printout that is easier to read and more useful to the parents and customers of Trusted Friends Daycare.



PROSKILLS

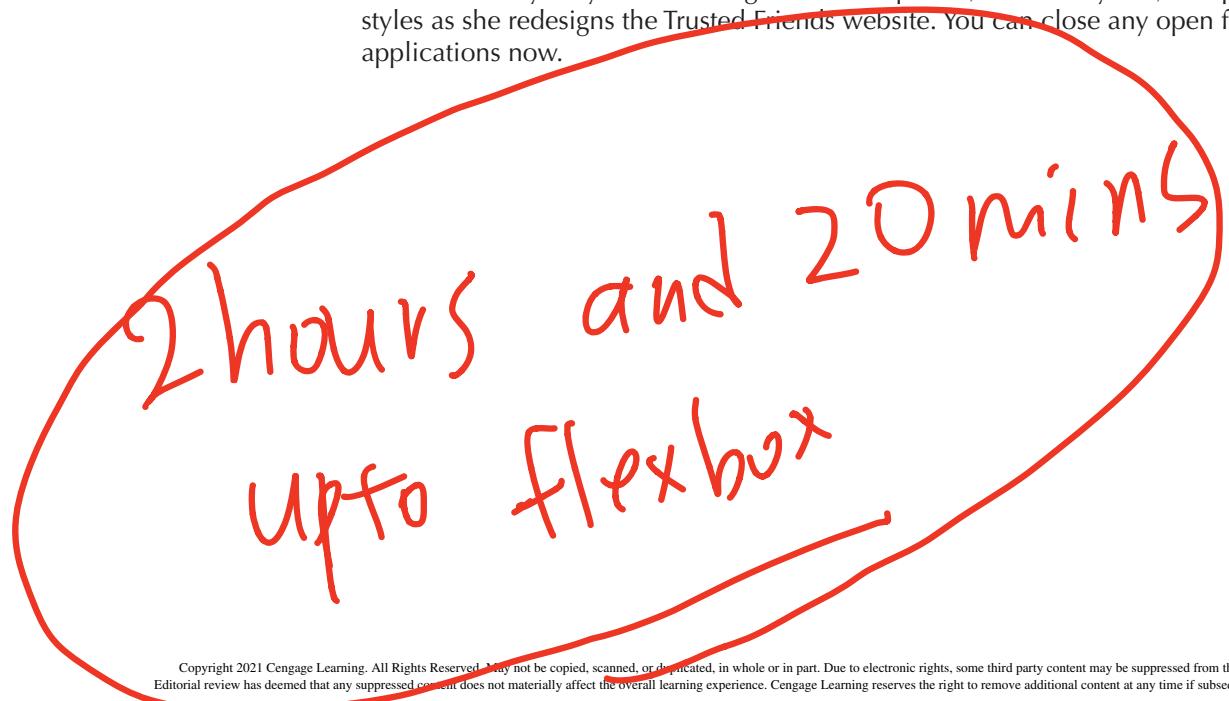
Written Communication: Tips for Effective Printing

One challenge of printing a web page is that what works very well on the screen often fails when transferred to the printed page. For example, some browsers suppress printing background images, so that white text on a dark background, which appears fine on the computer monitor, is unreadable when printed. Following are some tips and guidelines you should keep in mind when designing the printed version of your web page:

- *Remove the clutter.* A printout should contain only information that is of immediate use to the reader. Page elements such as navigation lists, banners, and advertising should be removed, leaving only the main articles and images from your page.
- *Measure for printing.* Use only those measuring units in your style sheet that are appropriate for printing, such as points, inches, centimeters, and millimeters. Avoid expressing widths and heights in pixels because those can vary with printer resolution.
- *Design for white.* Because many browsers suppress the printing of background images and some users do not have access to color printers, create a style sheet that assumes black text on a white background.
- *Avoid absolute positioning.* Absolute positioning is designed for screen output. When printed, an object placed at an absolute position will be displayed on the first page of your printout, potentially making your text unreadable.
- *Give the user a choice.* Some readers will still want to print your web page exactly as it appears on the screen. To accommodate them, you can use one of the many JavaScript tools available on the web that allows readers to switch between your screen and print style sheets.

Finally, a print style sheet is one aspect of web design that works better in theory than in practice. Many browsers provide only partial support for the CSS print styles, so you should always test your designs on a variety of browsers and browser versions. In general, you will have the best results with a basic style sheet rather than one that tries to implement a complicated and involved print layout.

In this tutorial you've learned how to apply different styles to different types of devices and output formats. Marjorie appreciates the work you've done and will continue to rely on your knowledge of media queries, flexible layouts, and print styles as she redesigns the Trusted Friends website. You can close any open files or applications now.

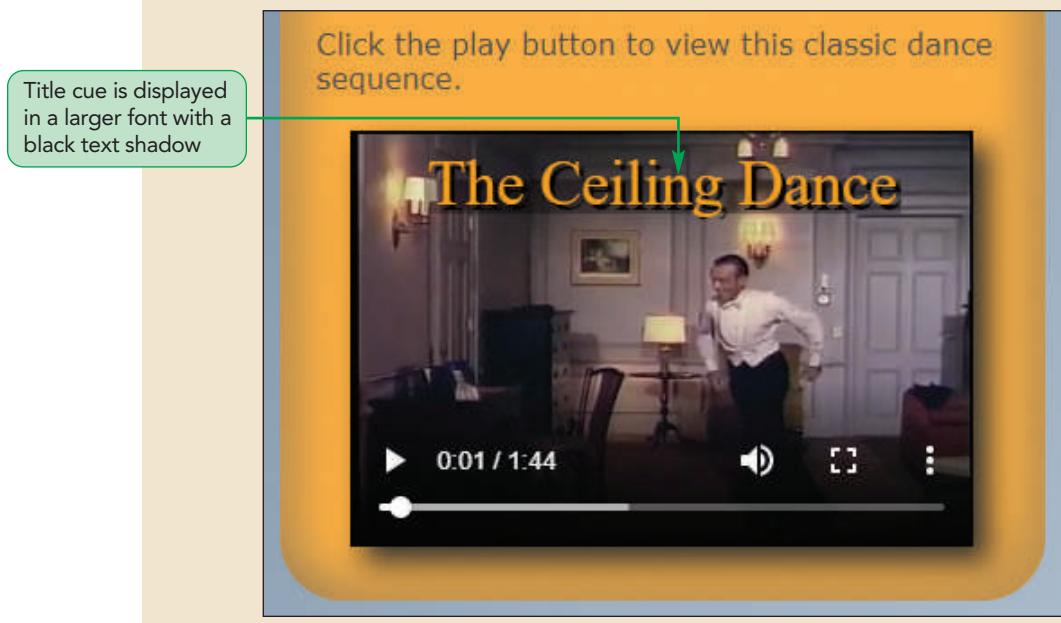


Session 5.3 Quick Check

1. What attribute do you add to a `link` element to indicate that the style sheet is used for printed media?
 - a. `rel="print"`
 - b. `type="print"`
 - c. `media="print"`
 - d. `print="yes"`
2. What `@rule` is used for setting the properties of the printed page box?
 - a. `@page`
 - b. `@print`
 - c. `@margin`
 - d. `@printout`
3. To set the right-side printed page to have a 3 centimeter top/bottom margin and a 5 centimeter left/right margin, use:
 - a. `@page:right {margin: 3cm 5cm;}`
 - b. `@page:right {margin: 5cm 3cm;}`
 - c. `@page {side: right; margin: 5cm 3cm;}`
 - d. `@page.right {margin: 5cm 3cm;}`
4. To apply a page break before every `section` element, use:
 - a. `section {break: before;}`
 - b. `section {page-break: before;}`
 - c. `section {break-before: true;}`
 - d. `section {break-before: always;}`
5. To prevent a page break from being placed within any `header` element, use:
 - a. `header {break: never;}`
 - b. `header {page-break-inside: avoid;}`
 - c. `header {inside-break: never;}`
 - d. `header {break: none;}`
6. What style do you apply to allow the browser to wrap long strings of text to a new line whenever needed?
 - a. `word-break: auto;`
 - b. `word-wrap: true;`
 - c. `word-wrap: break-word;`
 - d. `word-inside-break: always;`
7. To limit the size of widows for all `article` elements to 3 lines or more, use:
 - a. `article {widows: 2;}`
 - b. `article {widows: 3;}`
 - c. `article {widows: 3+;}`
 - d. `article {widows: >2;}`
8. To display the URL of a hypertext link, use the property:
 - a. `attr(link)`
 - b. `attr(url)`
 - c. `attr(hypertext)`
 - d. `attr(href)`

Figure 8-28

Formatted text from the Title cue



Source: Archive.org

Trouble? At the time of this writing, support for video captions is mixed. Only Google Chrome and Opera support all of the text styles used to format captions. Safari supports styles for font size and Firefox supports styles for font colors. Edge does not support caption styling.

Maxine likes your work in creating and formatting the ceiling dance video clip. However, she would like to review other options for embedding multimedia content on her website.

Using Third-Party Video Players

Prior to the widespread adoption of HTML 5 for embedded video, browsers used plug-ins using the following `object` element

```
<object attributes>
  parameters
</object>
```

where `attributes` define the object and `parameters` are values passed to the object controlling the object's appearance and actions. The `object` element, which replaced the `embed` element introduced in the last session, could be used for any type of content—from sound and video clips to graphic images, PDF files, and even the content of other web pages.

The parameters of the `object` are defined using the following `param` element

```
<param name="name" value="value" />
```

where the `name` is the name of the parameter and the `value` is the parameter's value. There is no standard list of parameter names and values because they are based on the plug-in used to display the object. For example, the following `object` element could