TUTORIAL 2

Getting Started with CSS

Designing a Website for a Fitness Club

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

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

tutorial
tss_bike_txt.html
tss_home_txt.html
tss_run_txt.html
tss_swim_txt.html
tss_styles_txt.css
+ 8 files

review
tss_coach_txt.html
coach_styles_txt.css
+ 8 files

case1
ph_plays_txt.html
ph_styles_txt.css
+ 8 files

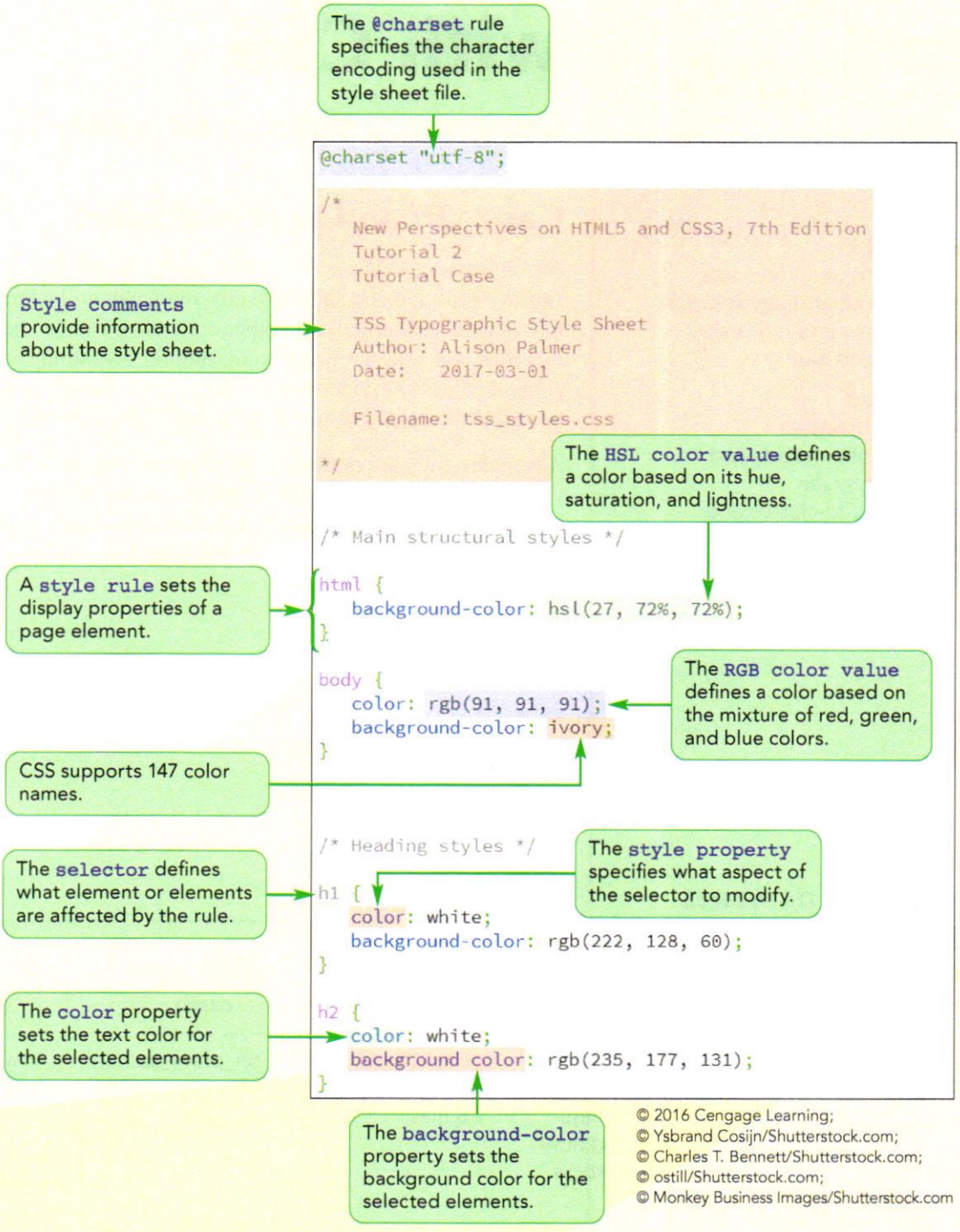
case2
mw_tour_txt.html
mw_styles_txt.css
+ 2 files

case3
cw_class_txt.html
cw_styles_txt.css
+ 3 files

case4
lake_home_txt.html
lake_styles_txt.css
+ 7 files

demo
demo_css.html
demo_color_names.html
+ 3 files

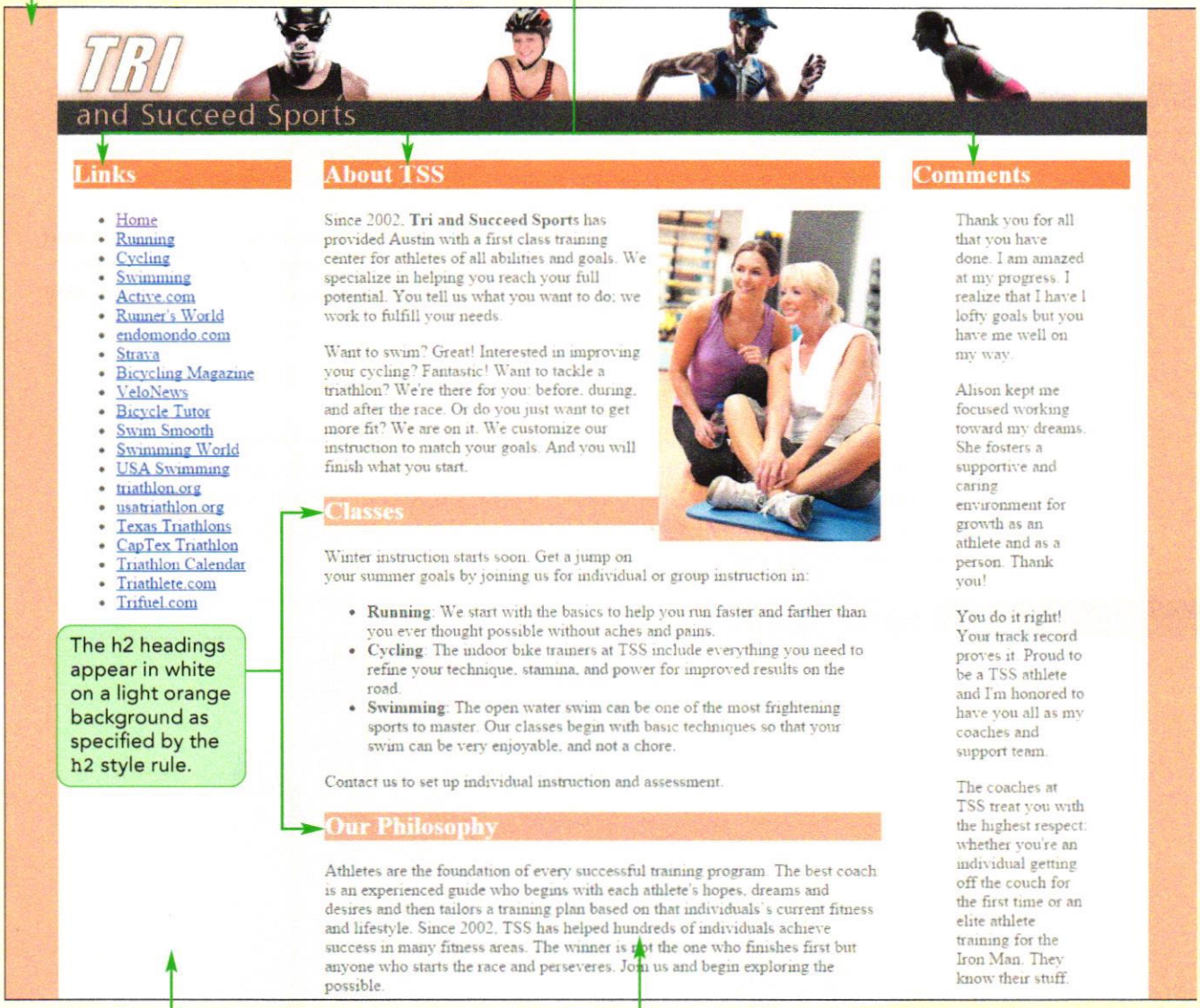
Session 2.1 Visual Overview:



CSS Styles and Colors

The browser window background color is set to the color value hsl(27, 73%, 72%) using the `html` style rule.

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



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

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

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

Introducing CSS

One of the important principles discussed in the previous tutorial was that HTML does not define how a document should be displayed; it only defines the document's structure and content. The appearance of the page is determined by one or more 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.

The CSS specifications are maintained by the same World Wide Web Consortium (W3C) that defines the standards for HTML. As with HTML, the CSS language has gone through several versions, the latest of which is CSS Version 3, more commonly known as **CSS3**. CSS3 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 CSS3 modules with each module experiencing a 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.

TIP

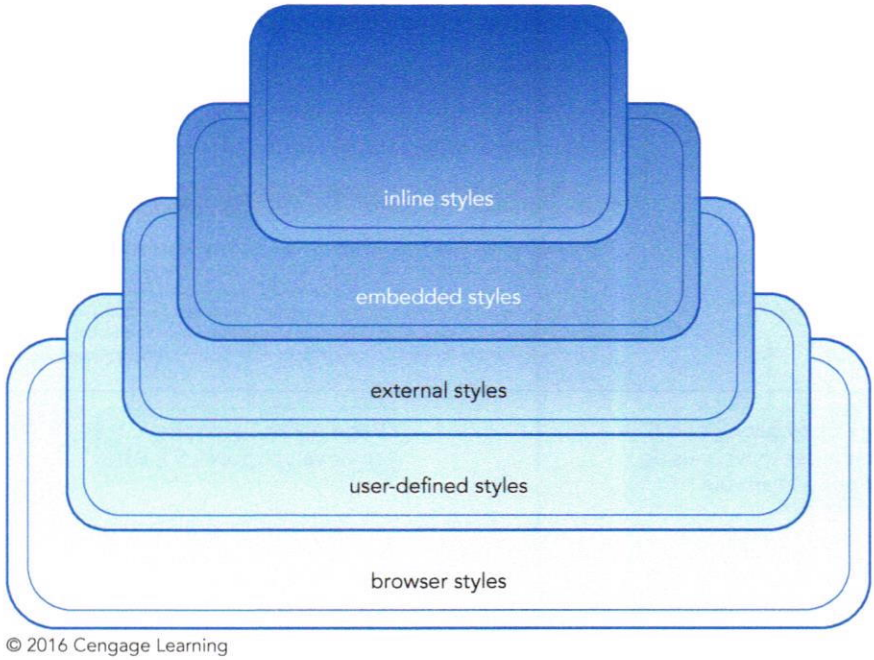
You can research the support for CSS by browser version at *www.caniuse.com*.

In these tutorials, 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 CSS designs.

Types of Style Sheets

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

Figure 2-1 Hierarchy of styles



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

The next styles to be processed are the **user-defined styles**, which are styles defined by the user based on the settings he or she makes in configuring the browser. For example, a user with a visual impairment could alter the browser's default settings to display text with highly contrasting colors and a large font size for improved readability. Any user-defined style has precedence over its browser style counterpart.

User-defined styles can be superseded by **external styles**, which are the styles that the website author creates and places within a CSS file and links to the page. 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, which makes it easy to apply a common design to an entire website.

Above the external styles in the hierarchy of style sheets are **embedded styles**, which are the styles added to the head of an HTML document. Embedded styles only apply to the HTML document in which they are created and they are not accessible to other documents in the website, but they do override any styles in an external style sheet.

Finally, at the highest order of precedence are **inline styles**, which are added as element attributes within an HTML document and thus apply to that element alone. Embedded styles and inline styles are not considered best practice and their use should be avoided because they violate the basic tenets of HTML, which is that HTML should only describe the content and structure of the document and that design styles should be placed outside of the HTML code.

The overall design of a web page 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.

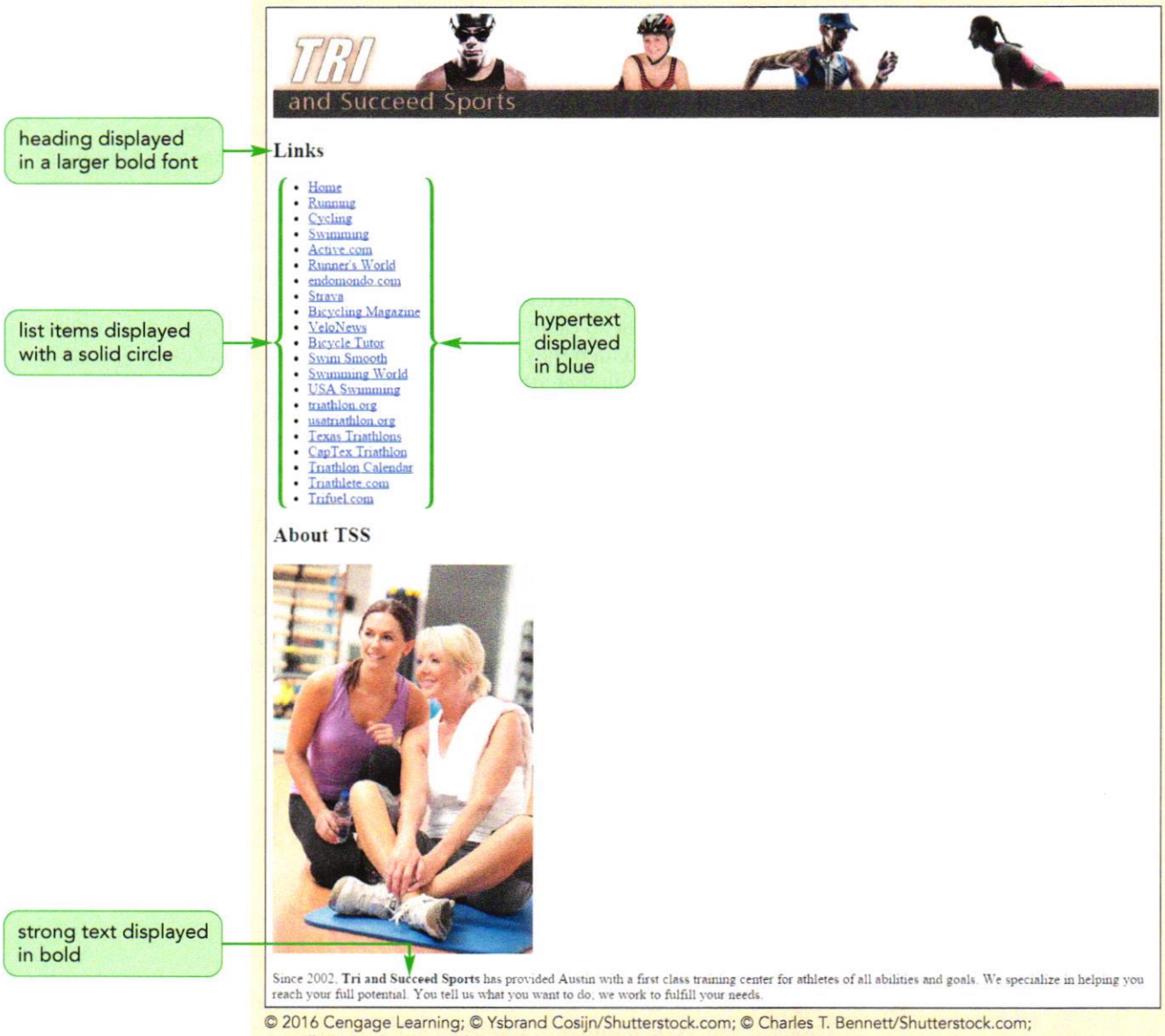
Viewing a Page Using Different Style Sheets

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

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



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 page layout is 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

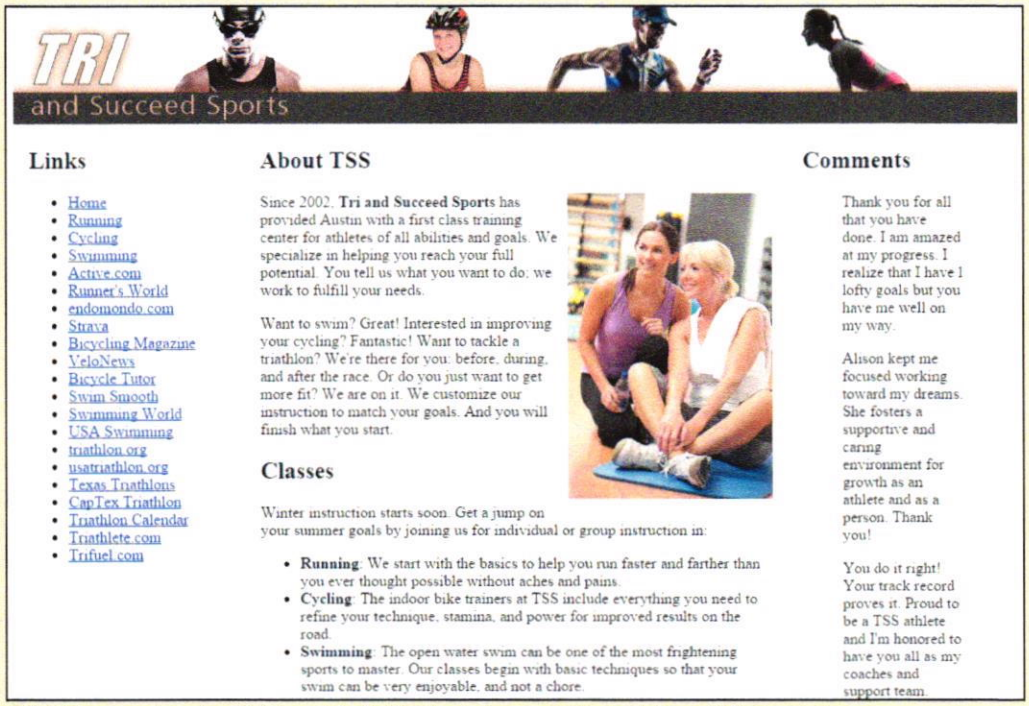
```
    <meta charset="utf-8" />
    <meta name="keywords" content="triathlon, running, swimming, cycling" />
    <title>Tri and Succeed Sports</title>
    <link href="tss_layout.css" rel="stylesheet" />
</head>
```

rel attribute indicates that the file is a style sheet

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



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:

```
h1 {
   color: red;
   text-align: center;
}
```

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

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

Like HTML, CSS ignores the use of white space, so you can also enter this style more compactly as follows:

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

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**, which indicates 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 CSS3 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 as metadata by adding the following `style` element to the document head

```
<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 changing designs tedious and inefficient. 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 pages and page elements.

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 specific 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. For example, 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 an element is rendered is that properties 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 that article. In addition, the text in a paragraph within that article is centered:

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

Thus, 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

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.

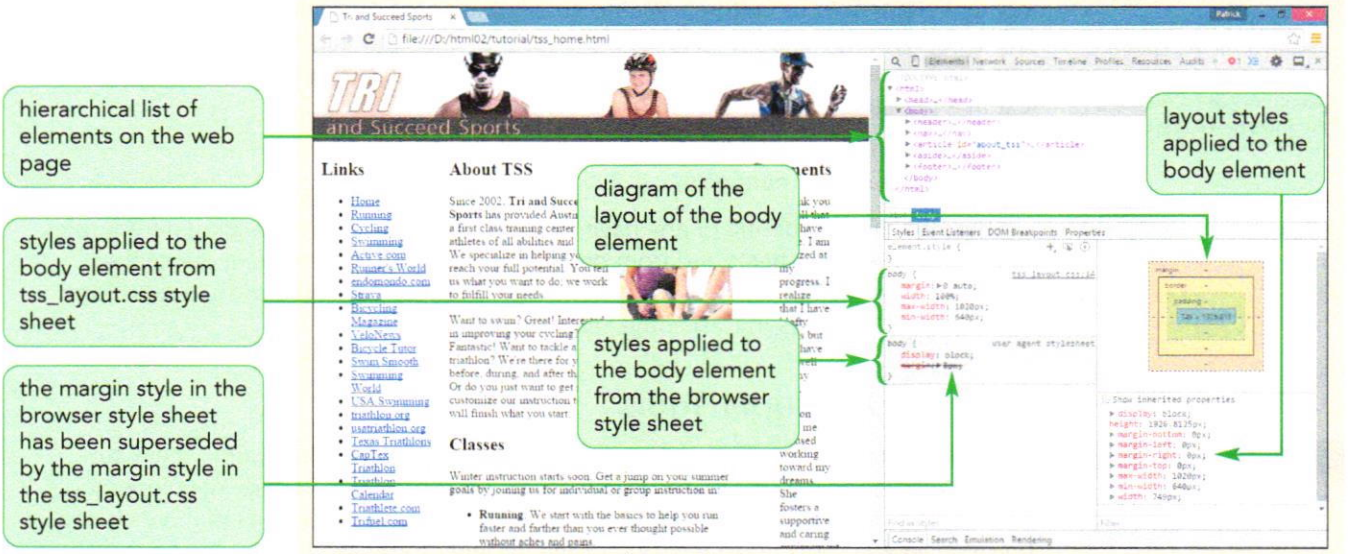
Accessing the Browser Developer Tools:

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+I or command+option+I.
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+I or by command+option+I.

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 permanently. 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";
```

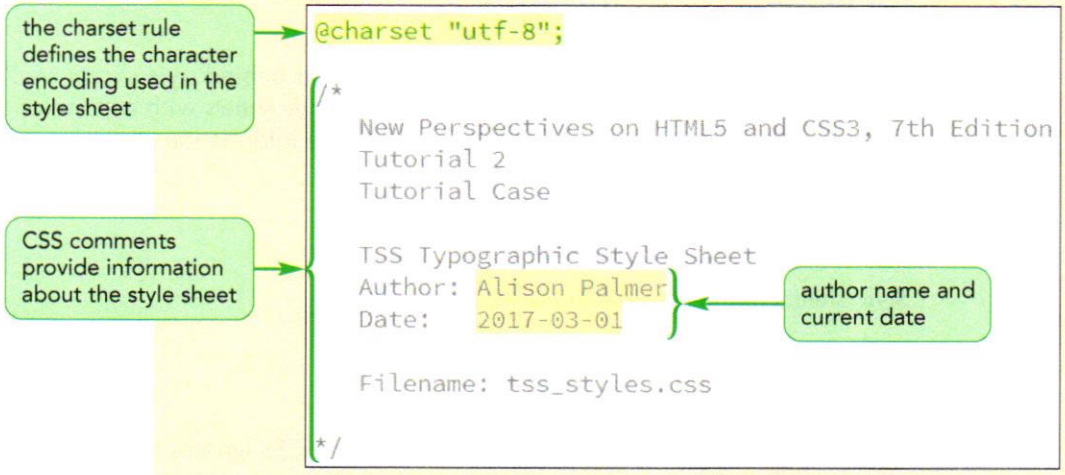
where *encoding* defines the character encoding used in the file. Add the `@charset` rule to the tss_styles.css style sheet file now, specifying that the UTF-8 character set is used in the CSS code.

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**



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(company.css);
@import url(support.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 either 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.

PROSKILLS

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

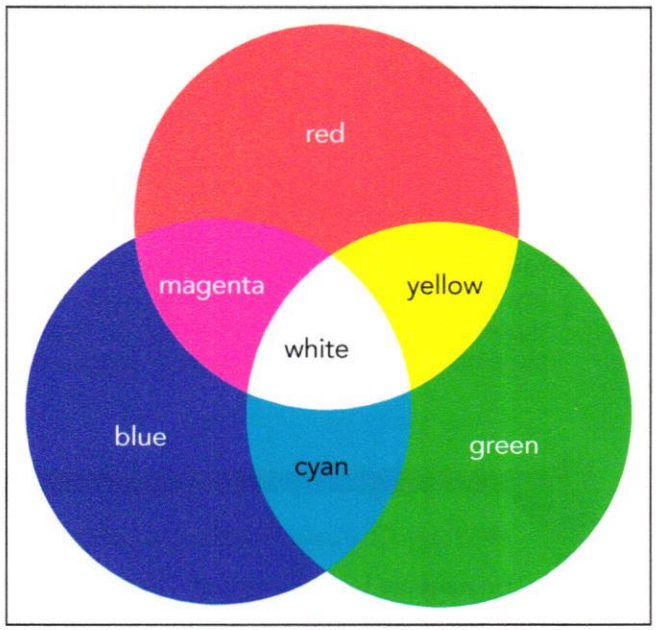
If your website will be used internationally, you need to be aware of how cultural differences can affect your audience's response to color. For instance, white, which is associated with innocence in Western cultures, is the color of mourning in China; yellow, which is considered a bright, cheerful color 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. CSS3 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)
```

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 256^3 (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
```

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

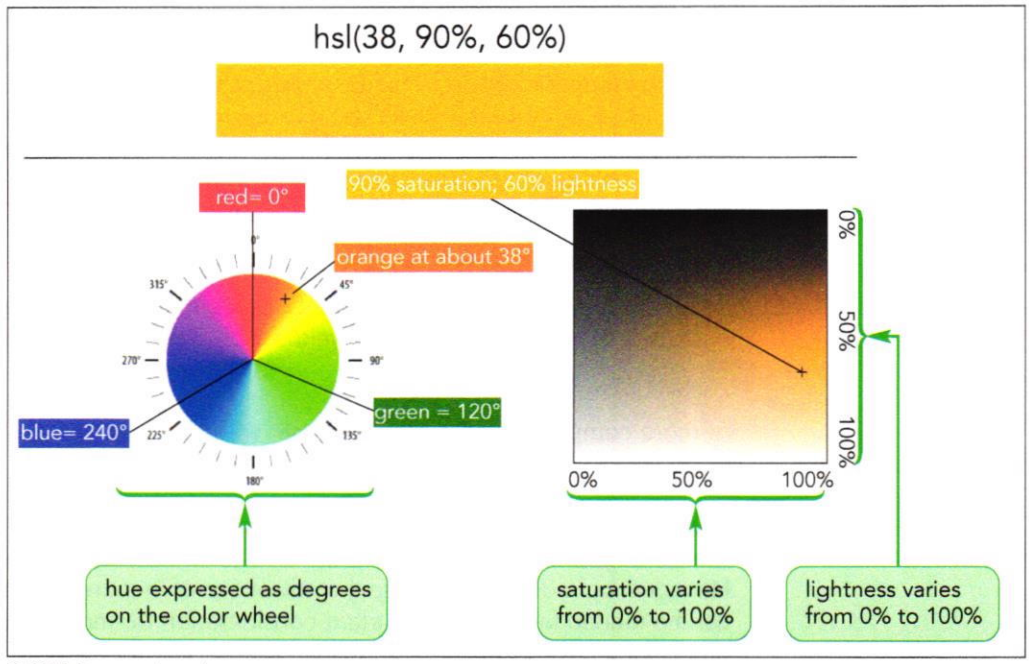
```
rgb(100%, 65%, 0%)
```

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

HSL Color Values

HSL color values were introduced in CSS3 and 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 CSS3 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%)
```

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

CSS3 introduced opacity to the CSS color models where **opacity** 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)
```

where *opacity* sets 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 background color. 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 are blended 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 to be 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.

To define background and text colors:

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

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

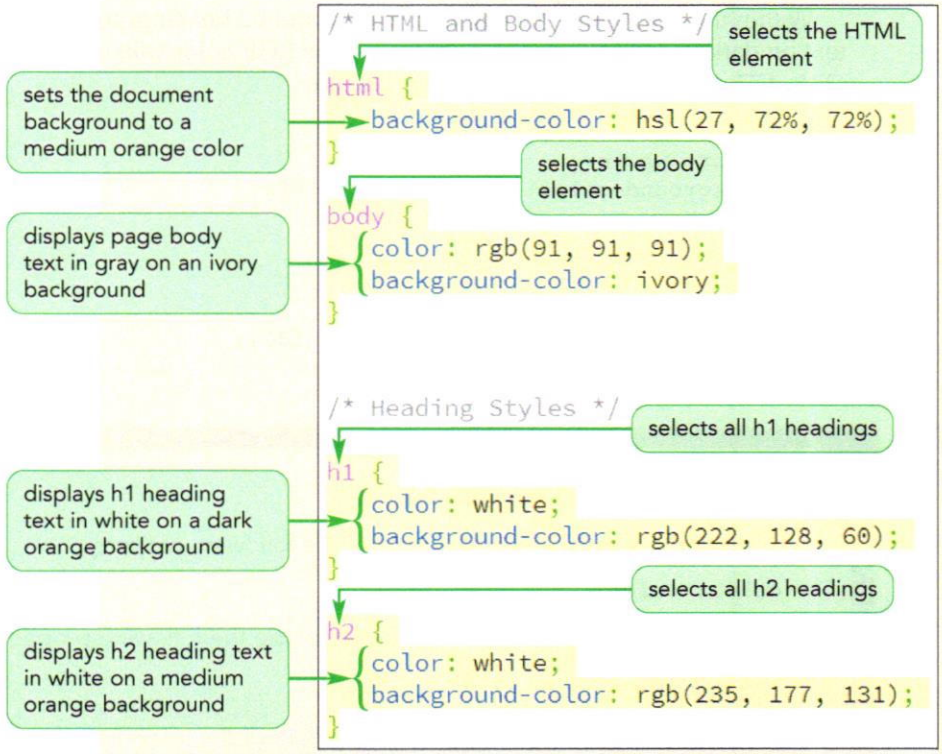
Almost 8% of all men and 0.5% of all women have some sort of color blindness. Because red-green color blindness is the most common type of color impairment, you should avoid using red text on a green background and vice-versa.

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**

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*—two hues close to one another on the color wheel in which one color is the dominant color and the other is a supporting color 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.

Employing Progressive Enhancement

The HSL color you used for the html selector was introduced with CSS3 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. What are inline styles, embedded styles, and external style sheets? Which would you use to define a design for an entire web site?
2. What keyword do you add to a style property to override style precedence and style inheritance?
3. Provide the code to enter the style comment "Tri and Succeed Sports Color Styles".
4. Provide the style rule to display block quote text in red using an RGB triplet.
5. The color chartreuse is located at 90° on the color wheel with 100% saturation and 50% lightness. Provide a style rule to display address text in black with chartreuse as the background color.
6. What is progressive enhancement?
7. Based on the following style rule for paragraph text, which style property will be used by an older browser that supports only CSS2?

```
p {
   color: rgb(232, 121, 50);
   color: hsla(23, 80%, 55%, 0.75);
}
```

8. Provide a style rule to display h1 and h2 headings with a background color of yellow (an equal mixture of red and green at highest intensity with no blue) at 70% opacity.

Session 2.2 Visual Overview:

The `@font-face` rule imports a web font into the style sheet.

The `font-family` property lists the possible fonts used for the element text.

The `font-size` property sets the text size in absolute or relative units.

The `letter-spacing` property sets the `kerning` or space between letters.

The `line-height` property sets the height of the lines of text in the element.

The `text-align` property sets the horizontal alignment of the text.

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

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

h1 {font-size: 2.2em;}
h2 {font-size: 1.5em;}

h1, h2 {
   font-family: Quicksand, Verdana, Geneva, sans-serif;
   letter-spacing: 0.1em;
}

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

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

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

The `em unit` is a relative unit of length that expresses a size relative to the font size of the containing element.

The `aside blockquote` selector selects `blockquote` elements that are descendants of the `aside` element.

The `nav > ul` selector selects `ul` elements that are direct children of the `nav` element.

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.

Body text is displayed in a Verdana font.

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 individuals'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 I 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!

TRI AND SUCCEED SPORTS • 41 VENTURE DR. • AUSTIN, TX 78711 • 512.555.9917

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.

Exploring Selector Patterns

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

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

This style rule will match an h1 heading located within a page article in the same way it matches an h1 heading nested within an `aside` element or the body header or the body footer. Often, however, you will want your style rules to apply to specific elements, such as h1 headings found within articles but not anywhere else. To direct a style rule to specific elements, you'll use **selector patterns** to match only those page elements that correspond to 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 matched. 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 within the `header` element:

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

As shown in the code that follows, 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>
```

In this example, 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
*	Matches any element
elem	Matches the element *elem* located anywhere in the document
elem1, elem2, …	Matches any of the elements *elem1*, *elem2*, etc.
parent descendant	Matches the *descendant* element that is nested within the *parent* element at some level
parent > child	Matches the *child* element that is a child of the *parent* element
elem1 + elem2	Matches *elem2* that is immediately preceded by the sibling element *elem1*
elem1 ~ elem2	Matches *elem2* that follows the sibling element *elem1*

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 are used to select elements based on 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:

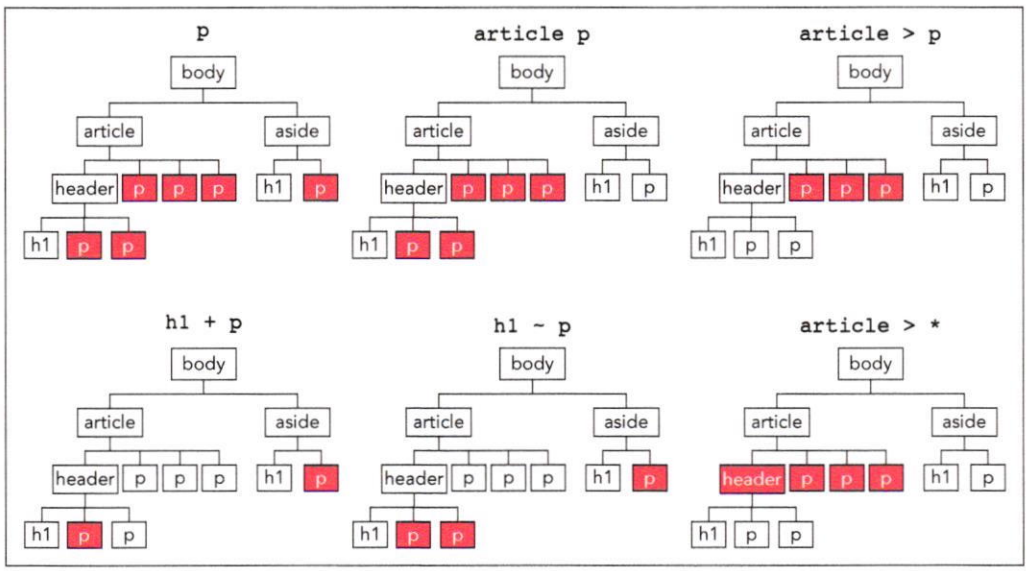
```
h1+h2 {color: blue;}
```

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.

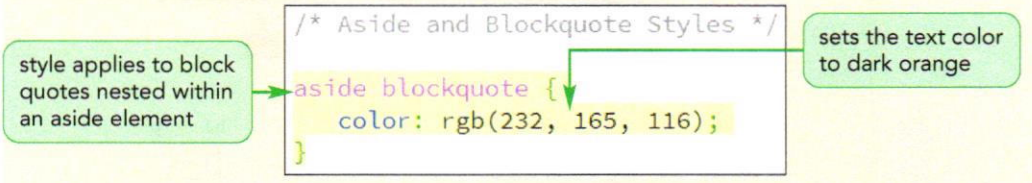
To create style rule with a contextual selector:

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**



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.

Attribute Selectors

Selectors also can be defined 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. For example, 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
elem#*id*	Element *elem* with the ID value *id*	h1#intro	The h1 heading with the id *intro*
#*id*	Any element with the ID value *id*	#intro	Any element with the id *intro*
elem.*class*	All *elem* elements with the class attribute value *class*	p.main	All paragraphs belonging to the *main* class
.*class*	All elements with the class value *class*	.main	All elements belonging to the *main* class
elem[*att*]	All *elem* elements containing the *att* attribute	a[href]	All hypertext elements containing the href attribute
elem[*att*="*text*"]	All *elem* elements whose *att* attribute equals *text*	a[href="top.html"]	All hypertext elements whose href attribute equals *top.html*
elem[*att*~="*text*"]	All *elem* elements whose *att* attribute contains the word *text*	a[rel~="glossary"]	All hypertext elements whose rel attribute contains the word *glossary*
elem[*att*|="*text*"]	All *elem* elements whose *att* attribute value is a hyphen-separated list of words beginning with *text*	p[id|="first"]	All paragraphs whose id attribute starts with the word *first* in a hyphen-separated list of words
elem[*att*^="*text*"]	All *elem* elements whose *att* attribute begins with *text* [***CSS3***]	a[rel^="prev"]	All hypertext elements whose rel attribute begins with *prev*
elem[*att*$="*text*"]	All *elem* elements whose *att* attribute ends with *text* [***CSS3***]	a[href$="org"]	All hypertext elements whose href attribute ends with *org*
elem[*att**="*text*"]	All *elem* elements whose *att* attribute contains the value *text* [***CSS3***]	a[href*="faq"]	All hypertext elements whose href attribute contains the text string faq

Note that some of the attribute selectors listed in Figure 2-15 were first introduced in CSS3 and, thus, might not be supported in older browsers.

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

selects h1 headings within an article element with the about_tss id

selects h2 headings within an article element with the about_tss id

```
/* 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);
}
```

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 instruction 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 before the closing `</head>` tag to link these files to the tss_layout.css and tss_styles.css files, respectively:

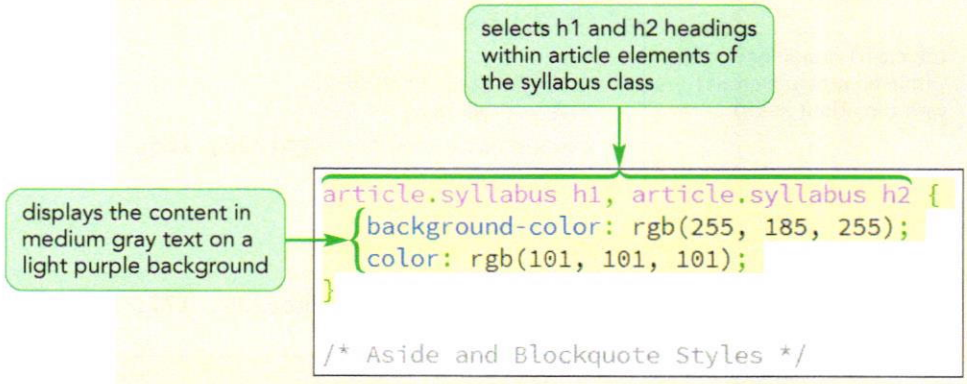
```
<link href="tss_layout.css" rel="stylesheet" />
<link href="tss_styles.css" rel="stylesheet" />
```

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);
}
```

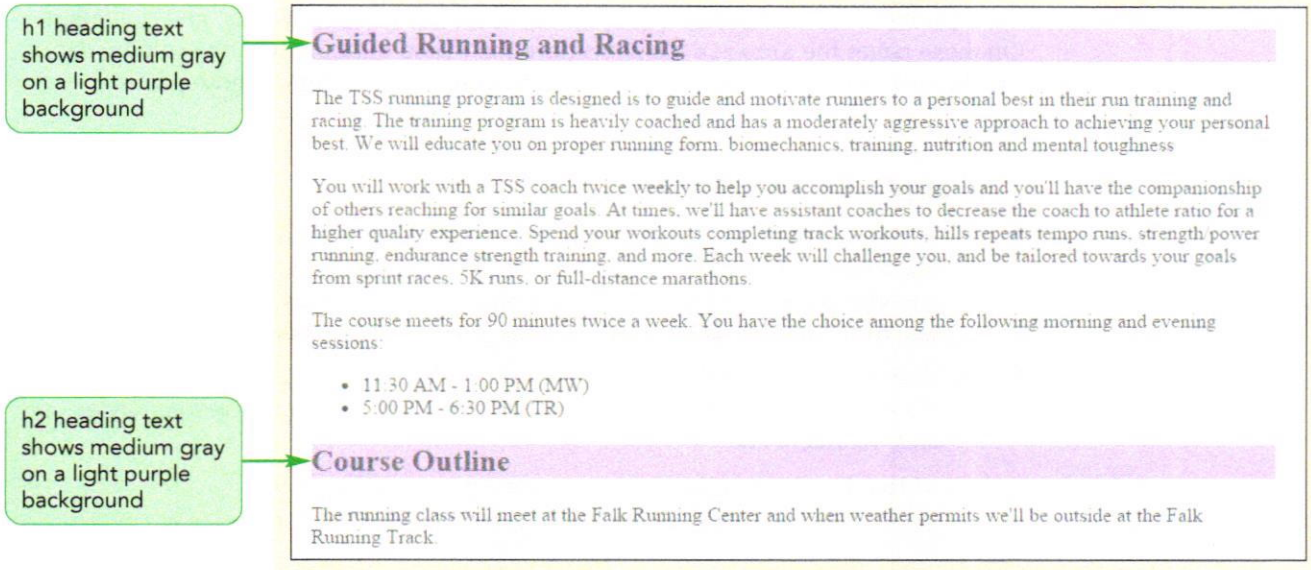
Figure 2-17 highlights the new style rule in the file.

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



7. Use the navigation links on the page to view the content and design of the cycling and the swimming pages, and then confirm that the h1 and h2 headings on these pages have similar formats.

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, every inline style has the value (1, 0, 0, 0) and thus 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, and end 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 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

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 applied 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.

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.

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

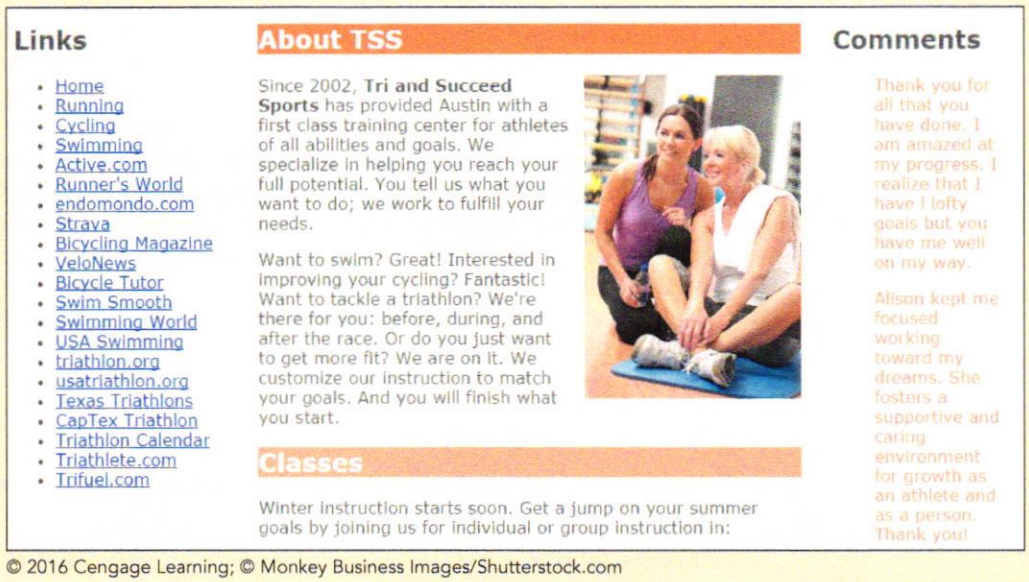
Figure 2-20 Specifying a font stack

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



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	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	Chrome, 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.

To install and use a web font:

1. Return to the **tss_styles.css** file in your editor.

TIP

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

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

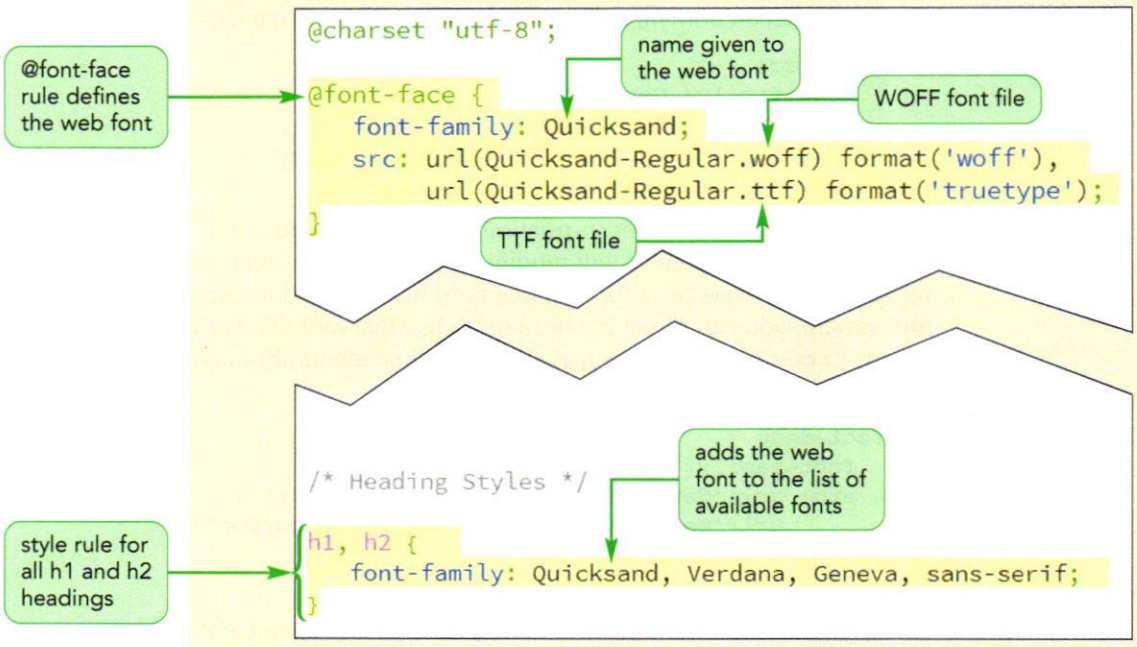
```
@font-face {
   font-family: Quicksand;
   src: url('Quicksand-Regular.woff') format('woff'),
        url('Quicksand-Regular.ttf') format('truetype');
}
```

3. At the top of the section for Heading Styles, 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

INSIGHT

Using Google Fonts

Google Fonts (*google.com/fonts*) hosts a library of free web fonts. Once you have selected fonts 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" />
```

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

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

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. For example, 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;}
```

TIP

You explore typography styles using the demo_css.html file from 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 this `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 rems, 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:

```
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. CSS3 introduced 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 add the following style rule to set the default font size of text in the `aside` element to 0.8em:

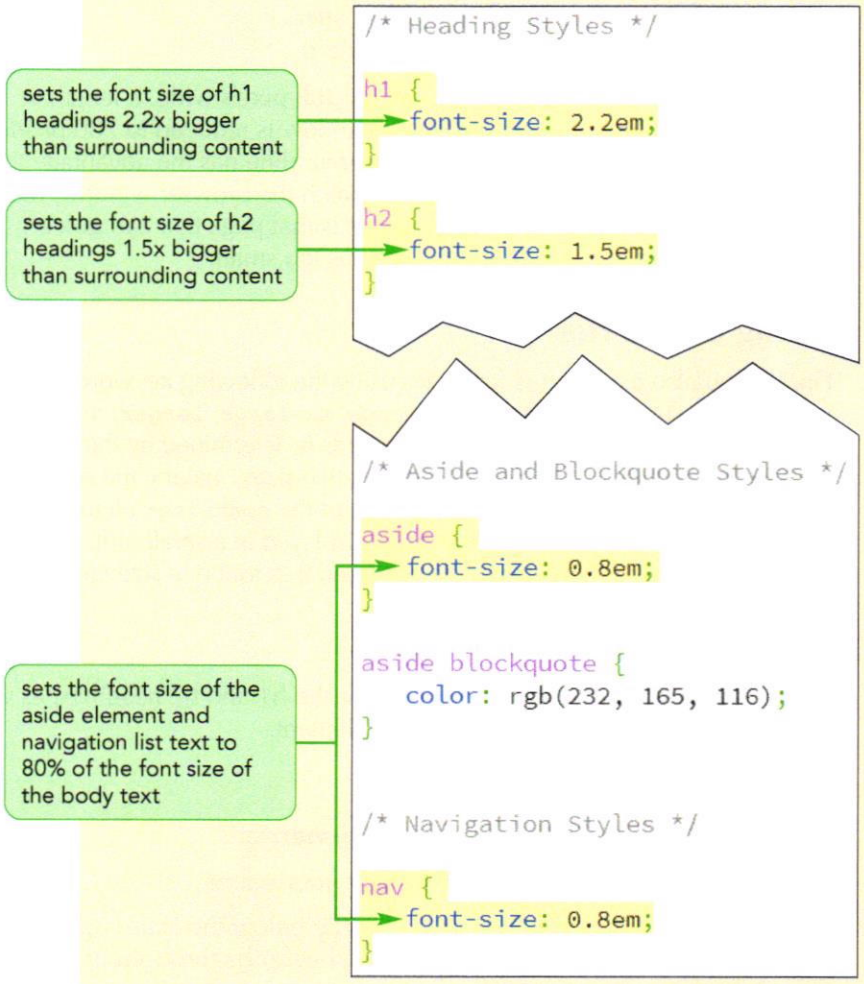
```
aside {
   font-size: 0.8em;
}
```

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

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

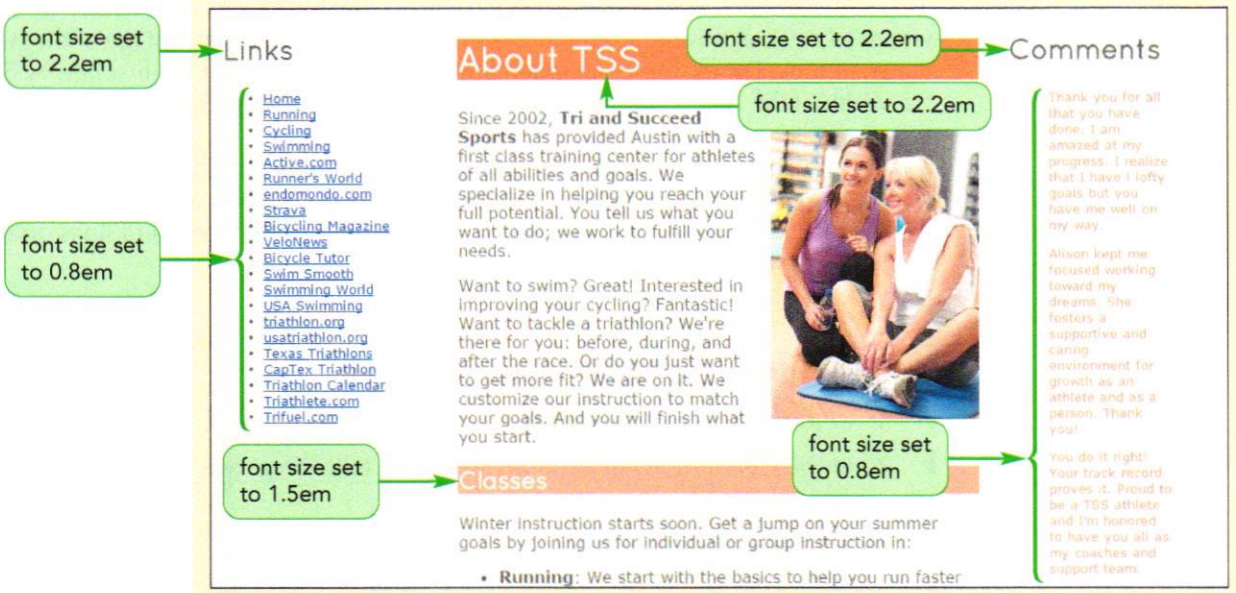
Figure 2-25 highlights the new font sizes for the website.

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 measures the amount of space between characters, while **tracking** measures the amount of 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 measures the amount of space between lines of text and is set using the following `line-height` property

```
line-height: size;
```

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;
```

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.

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 insert 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
`baseline`	Aligns the baseline of the element with the baseline of the parent element
`bottom`	Aligns the bottom of the element with the bottom of the lowest element in the line
`middle`	Aligns the middle of the element with the middle of the surrounding content in the line
`sub`	Subscripts the element
`super`	Superscripts the element
`text-bottom`	Aligns the bottom of the element with the bottom of the text in the line
`text-top`	Aligns the top of the element with the top of the text in the line
`top`	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.

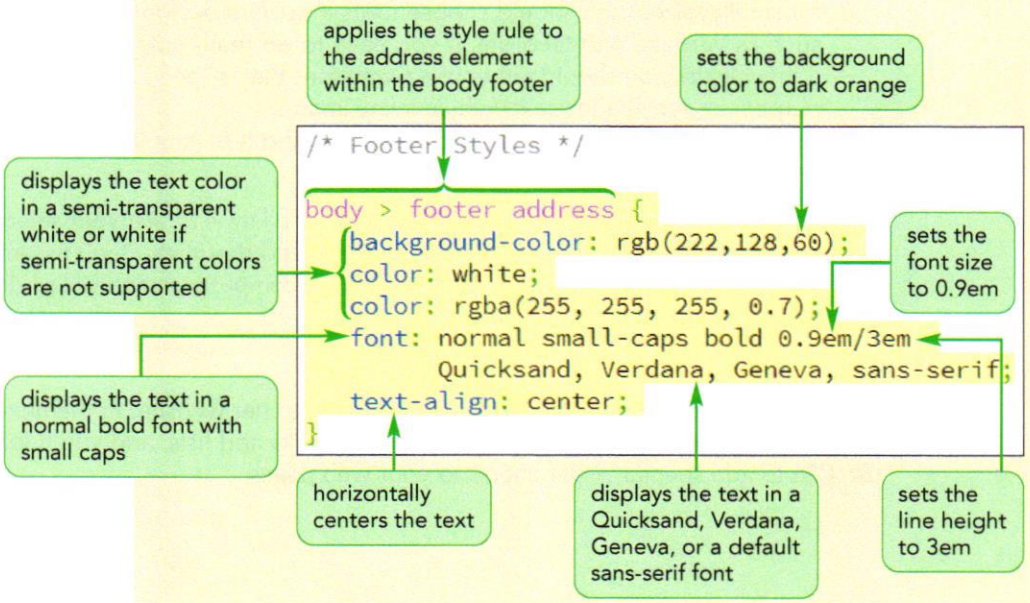
To apply the font property:

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;
}
```

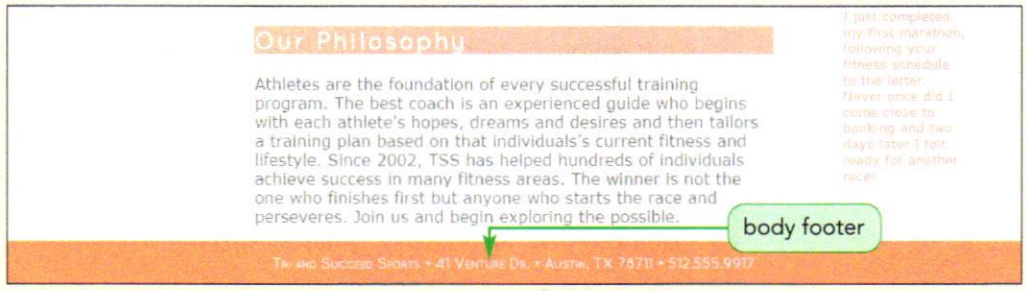
Note that this style rule uses progressive enhancement by placing each color rule on its own line so that browsers that do not support semi-transparent colors will display the address text in white. Figure 2-29 highlights the style rule for the footer.

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**



PROSKILLS

Decision Making: Selecting a Font

HTML and CSS provide a lot of typographic design 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 computer 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.

REVIEW

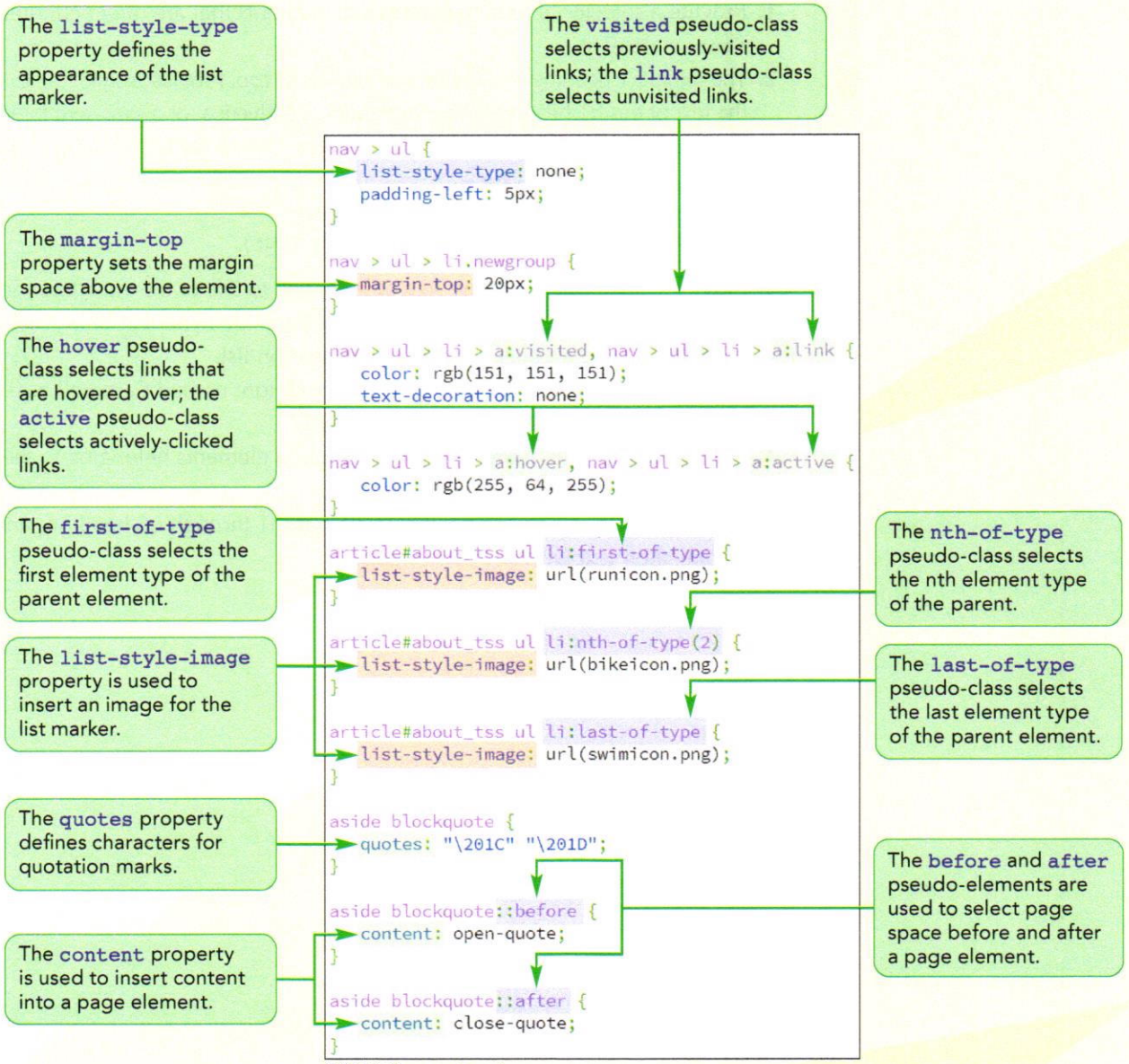
Session 2.2 Quick Check

1. Provide a selector to match all `address` elements that are direct children of the `footer` element.
2. The initial h1 heading in a document has the ID *top*. Provide a style rule to display the text of this h1 heading in Century Gothic, Helvetica, or a sans-serif font.
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;}
```

4. Provide a style rule to set the size of body text to 2% of the viewport width.
5. Provide a style rule to remove underlining from the hypertext links marked with the `<a>` tag and nested within a navigation list.
6. Provide the `@font-face` rule to create a web font named Cantarell based on the font files cantarell.woff and cantarell.ttf.
7. Provide a style rule to display all `blockquote` elements belonging to the Reviews class in italic and indented 3em.
8. Provide a style rule to horizontally center all h1 through h6 headings and to display their text with normal weight.

Session 2.3 Visual Overview:



Pseudo Elements and Classes

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

Open quote character is inserted using the content property.

Close quote character is inserted using the content property.

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.

Classes

This is the first li element.

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.

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

This is the last li element.

This is the second li element.

Comments

"Thank you for all that you have done. I am amazed at my progress. I realize that I have I 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!"

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

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

TIP

List style properties can be applied to individual items in a list, through the `li` element.

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:

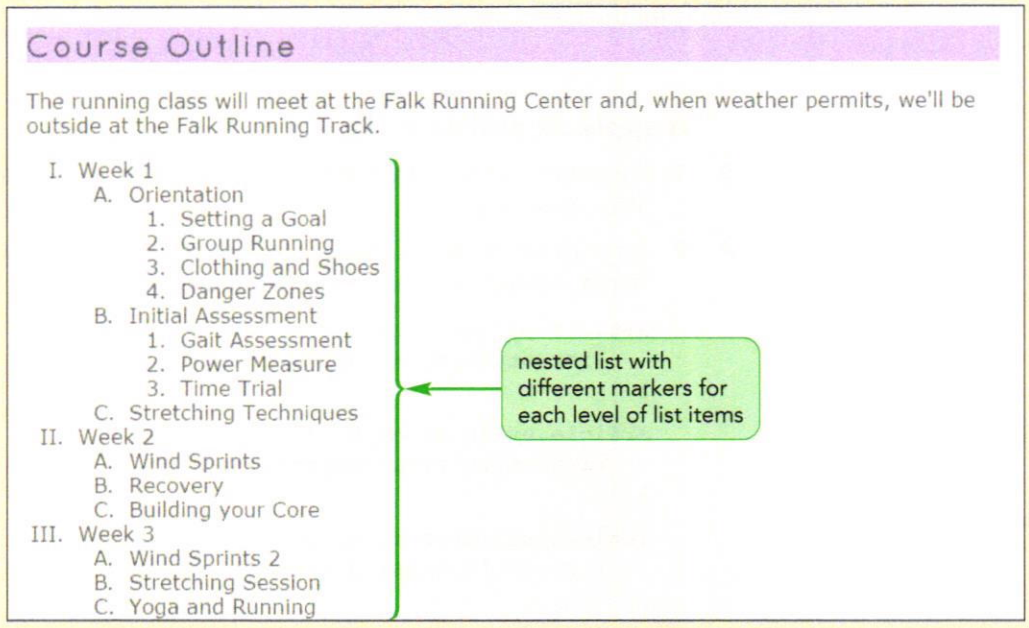
```
article.syllabus ol {
   list-style-type: upper-roman;
}

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 **Class outline**



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.

REFERENCE

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 using 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.

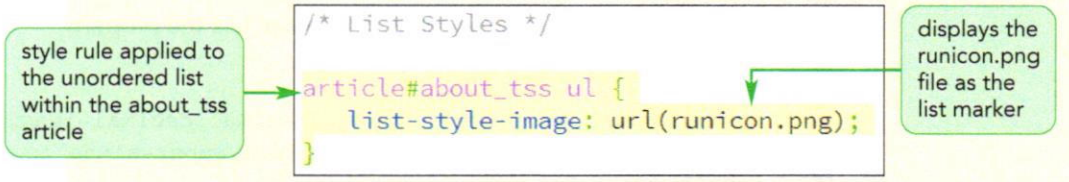
To use an image for a list marker:

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);
}
```

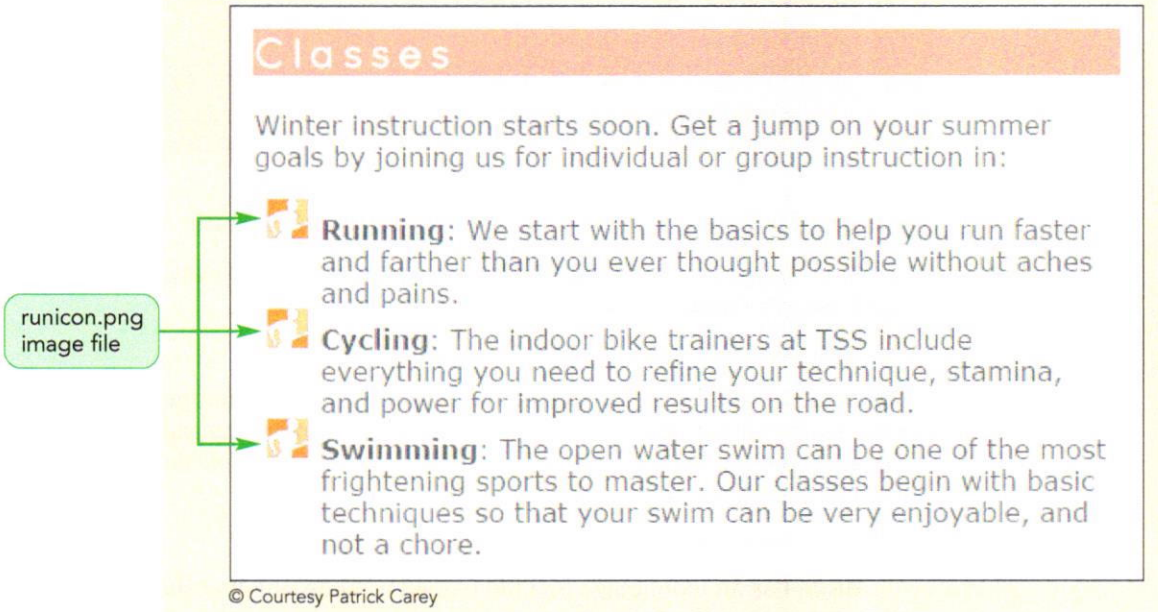
Figure 2-35 highlights the style rule to use the runicon.png file as the list marker image.

Figure 2-35 **Displaying an image in place of a list marker**



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**



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.

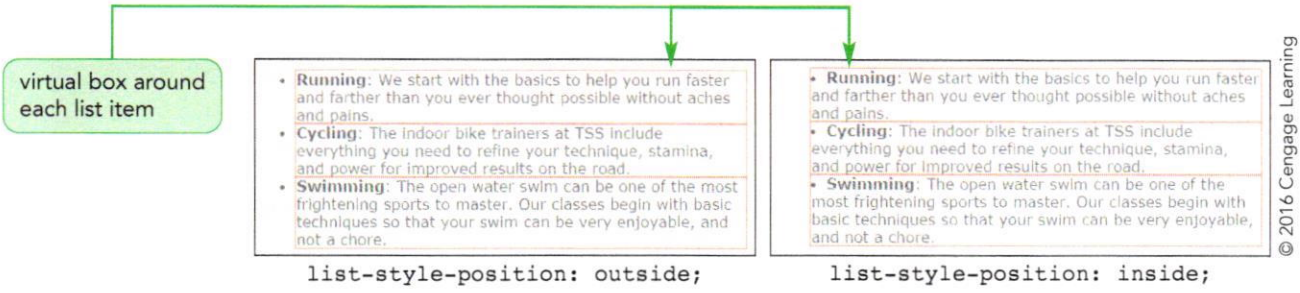
Setting the List Marker Position

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 concentric boxes:

- the content of the element itself
- the **padding space**, which extends from the element's content to a border
- the **border** surrounding the padding space
- the **margin space** comprised of the space beyond the border up to the next page element

Figure 2-38 shows a schematic 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. Likewise, two values set the padding spaces for the top/bottom edges and then 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;}
```

If you insert 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;}
```

If you want to define the padding space for one edge but not for the others, you can 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 other three remaining edges:

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

With ordered and unordered lists, the default style used by most browsers is to set 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.

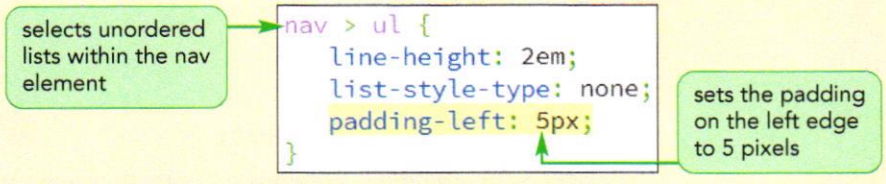
Include the unit in any style involving padding or margin spaces.

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 insert 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



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.

REFERENCE

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 have the same form as styles to set the padding space. To set the size of the margin around your block-level 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.

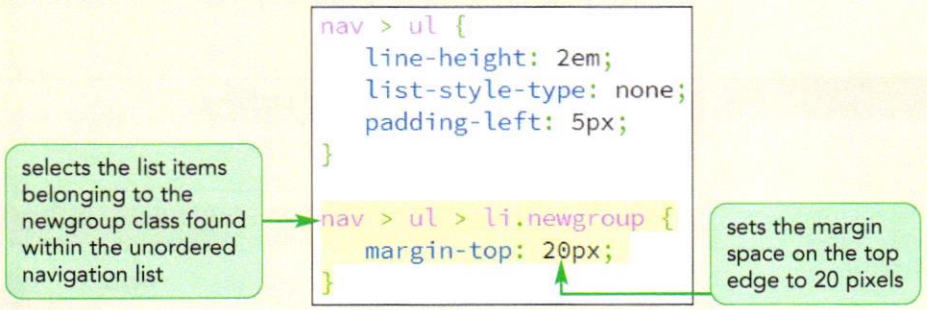
To increase the top margin:

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, insert 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



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.

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

selects block quotes within the aside element

```
aside blockquote {
   color: rgb(232, 165, 116);
   margin: 20px 5px;
}
```

sets the size of the top and bottom margins to 20 pixels

sets the size of the left and right margins to 5 pixels

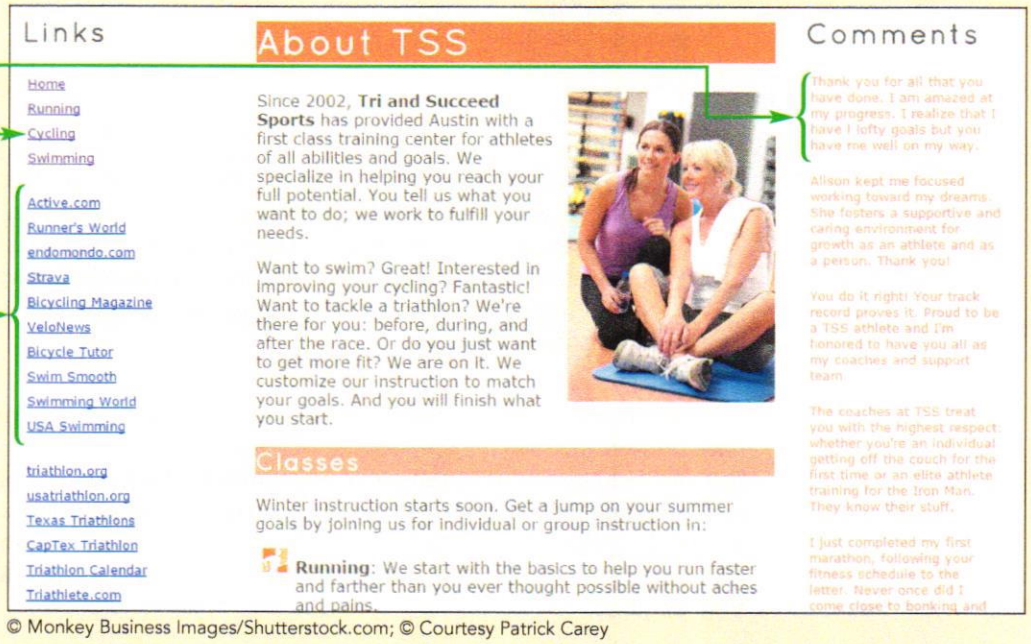
3. Save your changes to the file and then reload the **tss_home.html** file in your browser. Figure 2-42 displays the revised appearance of the page with the new padding and margin sizes applied to the navigation list and the block quotes.

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



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 up 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 is entered using 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
`:root`	The top element in the document hierarchy (the `html` element)
`:empty`	An element with no content
`:only-child`	An element with no siblings
`:first-child`	The first child of the parent element
`:last-child`	The last child of the parent element
`:first-of-type`	The first descendant of the parent that matches the specified type
`:last-of-type`	The last descendant of the parent that matches the specified type
`:nth-of-type(n)`	The n^{th} element of the parent of the specified type
`:nth-last-of-type(n)`	The n^{th} from the last element of the parent of the specified type
`:only-of-type`	An element that has no siblings of the same type
`:lang(code)`	The element that has the specified language indicated by *code*
`:not(selector)`	An element not matching the specified *selector*

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 maker. 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:

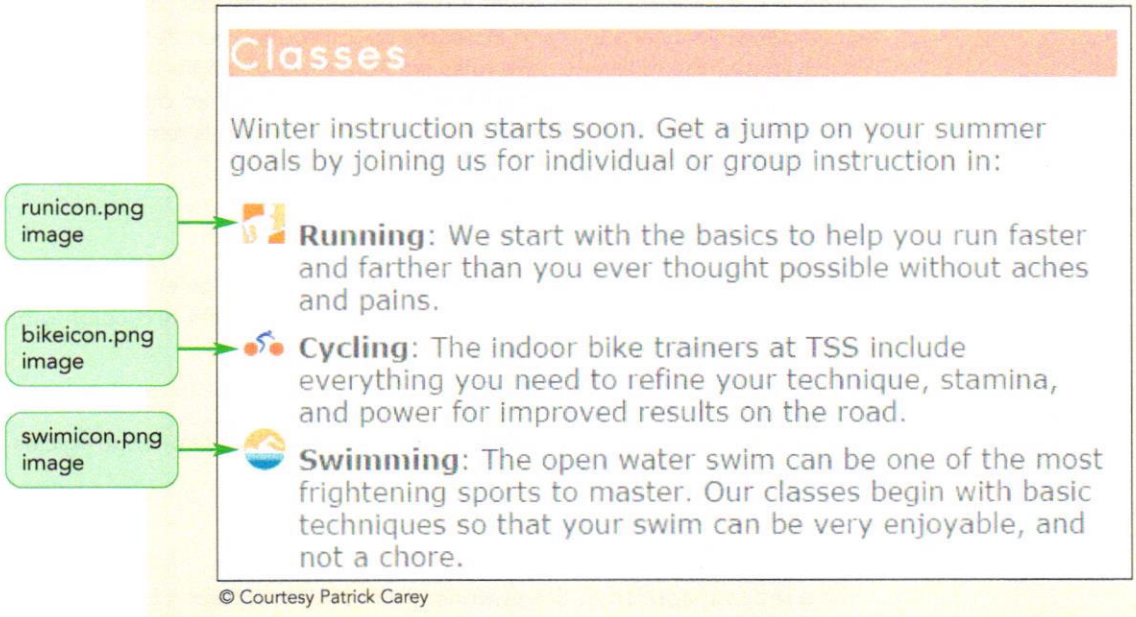
```
article#about_tss ul li:first-of-type {
   list-style-image: url(runicon.png);
}

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**



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 5th 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
`:link`	The link has not yet been visited by the user.
`:visited`	The link has been visited by the user.
`:active`	The element is in the process of being activated or clicked by the user.
`:hover`	The mouse pointer is hovering over the element.
`:focus`	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 insert the following style rules for hypertext links that have been visited or not visited.

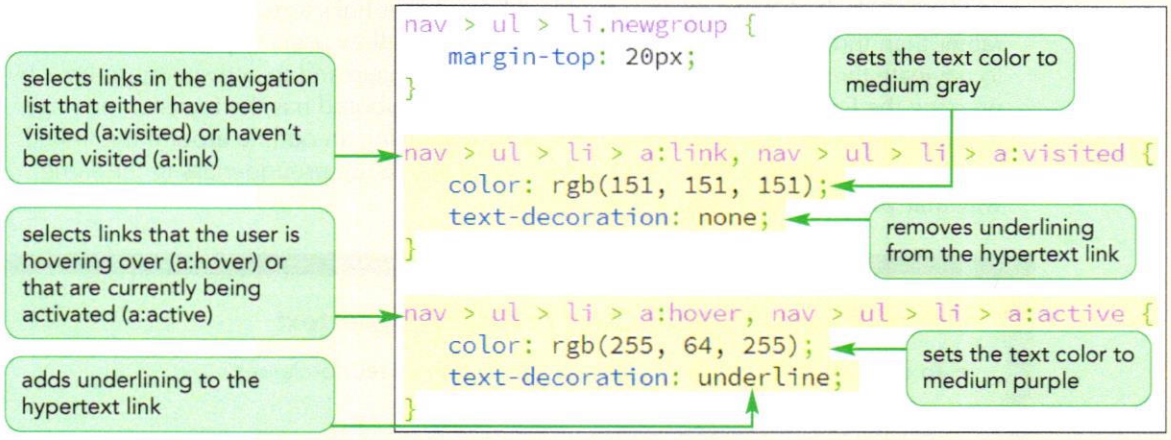
```
nav > ul > li > a:link, nav > ul > li > a:visited {
   color: rgb(151, 151, 151);
   text-decoration: none;
}
```

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. You'll explore how to work with this technique to create hidden menus on mobile devices in Tutorial 5.

With the increasing importance of touch devices, a good guiding principle is that you should avoid making support for the hover style a necessary condition 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
`::first-letter`	The first letter of the element text
`::first-line`	The first line of the element text
`::before`	Content inserted directly before the element
`::after`	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%;
}
```

Note that the double colon separator "`::`" was introduced in CSS3 to differentiate pseudo-elements from pseudo-classes. Older browsers use the singe 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(attribute)`	Displays the value of the selector's *`attribute`*
`text`	Displays the specified *`text`*
`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(url)`	Displays the content of the media (image, video, etc.) from the file located at *`url`*

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:

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

TIP

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

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 interpreting 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

```
<a href="http://www.triathlon.org">Triathlons</a>
```

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. You'll explore this issue further in Tutorial 5.

REFERENCE

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";
```

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 ’.

TIP

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

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, insert the following `quotes` property to use curly quotes for the quotation marks:

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

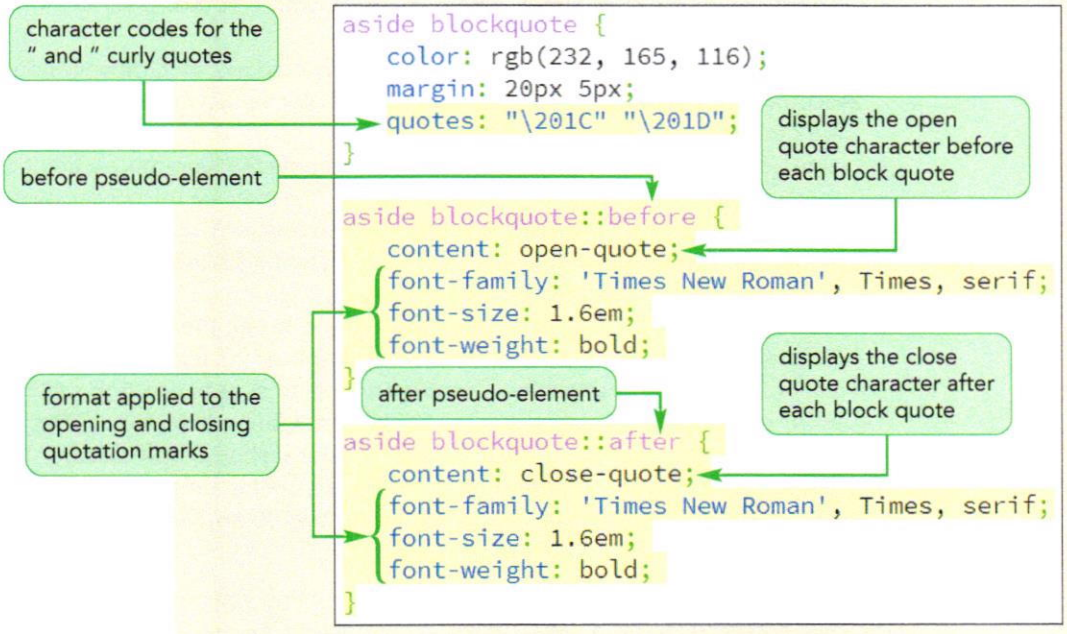
3. Add the following style rules to insert quotation marks before and after each block quote in the `aside` element:

```
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;
}
```

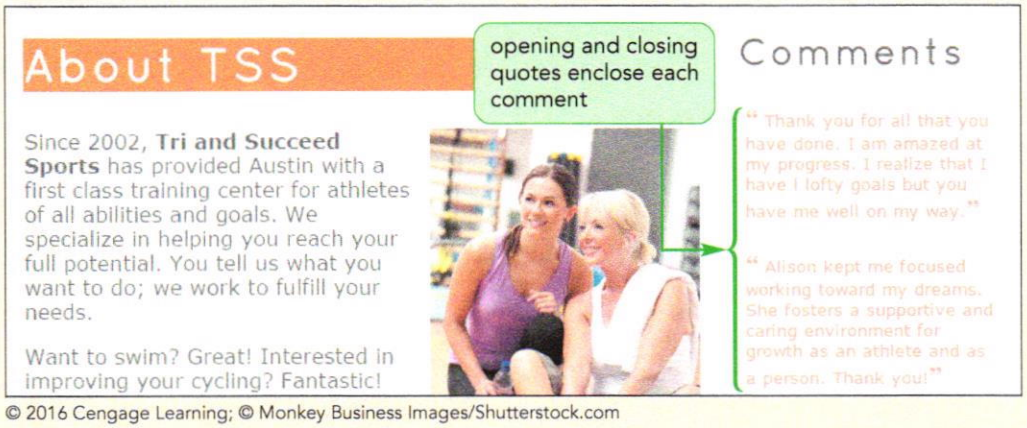
Figure 2-51 highlights the styles to add curly quotes before and after each block quote.

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



PROSKILLS

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.

REVIEW

Session 2.3 Quick Check

1. Provide a style rule to display all unordered lists with lowercase letters as the list marker.
2. Provide a style rule to display all unordered lists using the star.png image file, placed inside the virtual box.
3. Provide a style rule to display the text of all previously visited hypertext links in gray.
4. Provide the style rule to set the padding around every h1 heading in a `section` element to 1em on top, 0.5em on the left and right, and 2em on the bottom.
5. Provide the style rule to change the left margin of the `figure` element to 20 pixels.
6. Describe the item selected by the following selector:

```
#top > p:first-of-type:first-line
```

7. Describe the items selected by the following selector:

```
div.Links img[usemap]
```

8. Provide a style rule to insert the text string "***" before every paragraph belonging to the Review class.
9. Provide the style property to set the opening quotation mark and closing quotation marks to curly quotes with Unicode values of 2018 and 2019 respectively.

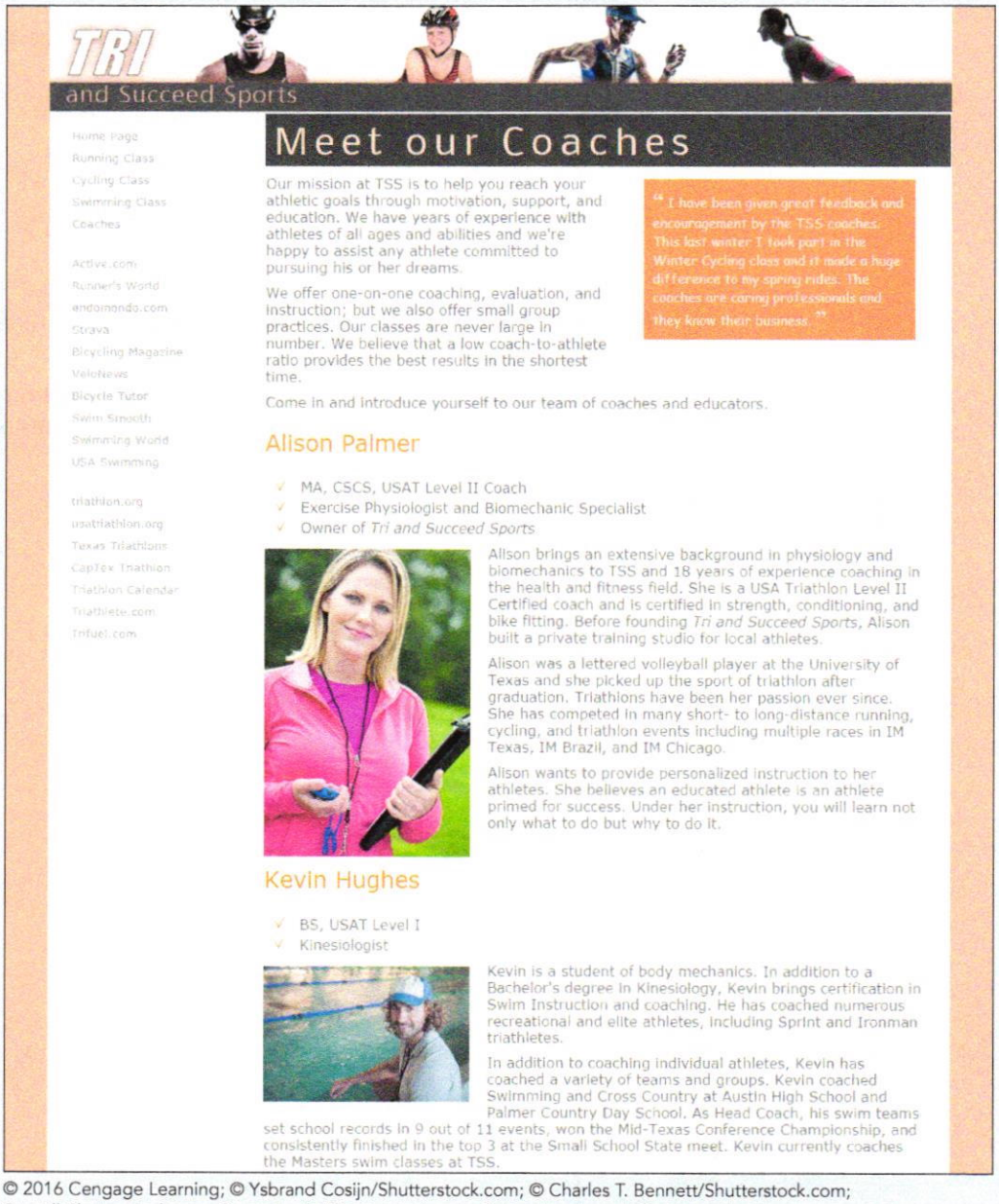
PRACTICE

Review Assignments

Data Files needed for the Review Assignments: coach_styles_txt.css, tss_coach_txt.html, 1 CSS file, 5 PNG files, 1 TTF file, 1 WOFF file

Alison has created another page for the Tri and Succeed Sports website providing biographies of the coaches at the club. She has already written the page content, acquired image files, and created a style sheet for the page layout. She wants you to finish the design of the page by developing a style sheet for the page's color scheme and typography. A preview of the page you'll design is shown in Figure 2-53.

Figure 2-53 TSS coaches profile page



Complete the following:

1. Use your HTML editor to open the **tss_coach_txt.html** and **coach_styles_txt.css** files from the html02 ▸ review folder. Enter ***your name*** and ***the date*** in the comment section of each file, and save them as **tss_coach.html** and **coach_styles.css** respectively.
2. Go to the **tss_coach.html** file in your editor and then within the document head, create links to the **coach_layout.css** and **coach_styles.css** style sheets.
3. Take some time to study the content and structure of the file and then close the document, saving your changes.
4. Go to the **coach_styles.css** file in your editor. At the top of the file and before the comment section do the following:
   a. Insert an `@charset` rule to set the character encoding for the file to utf-8.
   b. Use the `@font-face` rule to define a web font named Nobile, which is based on the nobile-webfont.woff file and, if that format is not supported, on the nobile-webfont.ttf file.
5. Go to the Main Structural Styles section and do the following:
   a. Change the background color of the browser window by creating a style rule for the `html` element that sets the background color to the value hsl(27, 72%, 72%).
   b. For the `body` element, create a style rule to set the text color to the value rgb(91, 91, 91), the background color to ivory, and body text to the font stack: Verdana, Geneva, sans-serif.
6. Create a style rule for the `body > footer address` selector containing the following styles:
   a. The background color set to the value rgb(222, 128, 60)
   b. The font color to white and then to the semitransparent value rgba(255, 255, 255, 0.6)
   c. The font style to normal displayed in bold small capital letters with a font size of 0.9em and a line height of 3em using the font stack Nobile, Verdana, Geneva, sans-serif
   d. The text horizontally centered on the page
7. Go to the Heading Styles section and create a style rule for every h1 heading that displays the text with a normal font weight from the font stack: Nobile, Verdana, Geneva, sans-serif. Set the letter spacing to 0.2em and the margin to 0 pixels.
8. Alison wants you to format the main h1 heading at the top of the page. Create a style rule for the `section#tss_coaches h1` selector that sets the font size to 2.5em with a color value of hsl(27, 82%, 85%) and background color of hsl(27, 6%, 21%). Set the left padding space to 10 pixels.
9. Alison also wants you to format the h2 headings for each coach. Create a style rule for the `article.coach_bio h2` selector that sets the font size to 1.6em with normal weight and the font color to rgb(240, 125, 0).
10. Alison has inserted a comment from an athlete about the coaches. Format this comment by going to the Blockquote Styles section and creating a style rule for the `aside blockquote` selector to do the following:
    a. Set the font size to 0.95em using the font stack 'Comic Sans MS', cursive.
    b. Set the font color to rgb(222, 128, 60) and use a semi-transparent background color with the value rgba(255, 2555, 255, 0.75).
    c. Set the padding space to 10 pixels.
    d. Define opening and closing quotes for the element using the Unicode character 201C and 201D respectively.
11. Format the appearance of the opening quotes by creating a style rule for the `aside blockquote::before` selector to write a boldfaced open quote before the block quote with the font size set to 1.6em from the font stack 'Times New Roman', Times, serif.
12. Format the appearance of the closing quotes by creating a style rule for the `aside blockquote::after` selector to write a boldfaced open quote after the block quote with the font size once again set to 1.6em from the font stack 'Times New Roman', Times, serif.
13. Next, you'll format the appearance of the navigation list by going to the Navigation Styles section and creating a style rule for `body > nav` selector that sets the text of the navigation list in a 0.8em font size with a line height of 2em.

14. Create a style rule for the `nav > ul` selector that removes the list marker and sets the left padding to 5 pixels.
15. Alison wants to break up the long list of links in the navigation list. Create style rules for the 6th and 16th `li` elements within the `nav > ul` selector that sets the size of the top margin of those items to 20 pixels.
16. For every previously visited or unvisited hypertext link within the `nav > ul > li` selector, set the text to the RGB color value rgb(151, 151, 151) and remove the underlining from the text link.
17. For every hovered or active hypertext link within the `nav > ul > li` selector, set the text color to RGB value rgb(222, 128, 60) and underline the hypertext link.
18. Go to the Paragraph Styles section and insert a style rule that sets the top margin and bottom margin to 10 pixels, the right margin to 30 pixels, and the left margin to 0 pixels for every paragraph in the document.
19. Every coach has a list of accomplishments. Go to the List Styles section and insert a style rule for the `article.coach_bio > header > ul` selector that displays the check.png file as the list marker and sets the margin space to 0 pixels, except for the bottom margin, which should be set to 10 pixels.
20. Save your changes to the style sheet and then open the **tss_coach.html** file in your browser. Verify that the color and typography match that shown in Figure 2-53. Verify that when you hover the mouse pointer over the links in the navigation list the text is displayed in an underlined orange font.

APPLY

Case Problem 1

Data Files needed for this Case Problem: ph_plays_txt.html, ph_styles_txt.css, 1 CSS file, 1 PNG file, 3 TTF files, 3 WOFF files

Philip Henslowe Classic Theatre Randall Chen is the media director for the *Philip Henslowe Classic Theatre*, a regional classical theatre in Coeur d'Alene, Idaho. You've been asked to work on the website design for the company. The first page you'll manage lists the plays for next summer's repertoire. A preview of the page is shown in Figure 2-54.

Figure 2-54 List of Plays at the Philip Henslowe Classic Theatre

The content and layout of the page has already been created for you. Your job will be to create a style sheet for the typography of the page.

Complete the following:

1. Using your editor, open the **ph_plays_txt.html** and **ph_styles_txt.css** files from the html02 ▸ case1 folder. Enter ***your name*** and ***the date*** in the comment section of each file, and save them as **ph_plays.html** and **ph_styles.css** respectively.

2. Go to the **ph_plays.html** file in your HTML editor, and within the document head create links to the **ph_layout.css** and **ph_styles.css** style sheet files. Take some time to study the content and structure of the document and then close the file, saving your changes.
3. Go to the **ph_styles.css** file in your editor, and at the top of the file before the comment section, define the character encoding used in the document as utf-8.
4. Randall has several web fonts that he wants used for the titles of the plays produced by the company. Add the following web fonts to the style sheet, using `@font-face` rules before the comment section:
   a. The Champagne font using the cac_champagne.woff and cac_champagne.ttf files
   b. The Grunge font using the 1942.woff and 1942.ttf files
   c. The Dobkin font using the DobkinPlain.woff and DobkinPlain.ttf files
5. Go to the Structural Styles section, creating a style rule that sets the background color of the `html` element to the value hsl(91, 8%, 56%).
6. Add a style rule for the `body` element to set the background color to the value hsl(58, 31%, 84%) and the font of the body text to the font stack: 'Palatino Linotype', 'Book Antiqua', Palatino, serif.
7. Create a style rule for the `header` element that sets the background color to black.
8. Create a style rule for every paragraph that sets the margin space to 0 pixels and the padding space to 5 pixels on top and 25 pixels on the right, bottom, and left.
9. For paragraphs that are direct children of the body element, create a style rule that sets the font size to 1.1em and horizontally centers the paragraph text.
10. Create a style rule for the address element that sets the font style to normal with a font size of 0.9em, horizontally centered on the page. Set the top and bottom padding to 10 pixels.
11. Next, you'll format the appearance of navigation lists on the page. Go to the Navigation Styles section and create a style rule for the `nav a` selector that displays the hypertext links using the font stack 'Trebuchet MS', Helvetica, sans-serif, and sets the top and bottom padding to 10 pixels.
12. For every unvisited and previously visited hypertext link within a `nav` element, set the text color to white, remove underlining from the link text, and set the background color to the semi-transparent value hsla(0, 0%, 42%, 0.4).
13. For every active or hovered link in a `nav` element, set the text color to the semi-transparent value hsla(0, 0%, 100%, 0.7) and set the background color to the semi-transparent value hsl(0, 0%, 42%, 0.7).
14. Go to the Section Styles section of the style sheet. In this section, you'll define the appearance of the four playbills. You'll start with the h1 headings from the sections. Create a style rule for the `section.playbill h1` selector that sets the font size to 3em and the font weight to normal. Set the margin space around the h1 headings to 0 pixels. Set the padding space to 20 pixels on top, 0 pixels on the right, 10 pixels on the bottom, and 20 pixels on the left.
15. Each playbill section is identified by a different ID value ranging from play1 to play4. Create style rules that set a different background color for each playbill using the following background colors:
    ID: play1 set to hsl(240, 100%, 88%)
    ID: play2 set to hsl(25, 88%, 73%)
    ID: play3 set to hsl(0, 100%, 75%)
    ID: play4 set to hsl(296, 86%, 86%)
16. Each playbill section heading will also have a different font. For the h1 headings within the four different playbills, create style rules to apply the following font stacks:
    ID: play1 set to Champagne, cursive
    ID: play2 set to Grunge, 'Times New Roman', Times, serif
    ID: play3 set to Impact, Charcoal, sans-serif
    ID: play4 set to Dobkin, cursive

17. Randall has put the author and the director of each play within a definition list. Format these definition lists now by going to the Definition List Styles section and creating a style rule for the `dt` element that sets the font size to 1.3em, the font weight to bold, and the font color to the semi-transparent value hsla(0, 0%, 0%, 0.4).
18. Create a style rule for every `dd` element to set the font size to 1.3em, the left margin space to 0 pixels, and the bottom margin space to 10 pixels.
19. Save your changes to the file and then open the **ph_plays.html** file in your browser. Verify that the typography and colors used in the document match those shown in Figure 2-54. Also, verify that, when you hover the mouse pointer over an item in the navigation lists for the entire page and for each play, the background color of the link becomes more opaque.

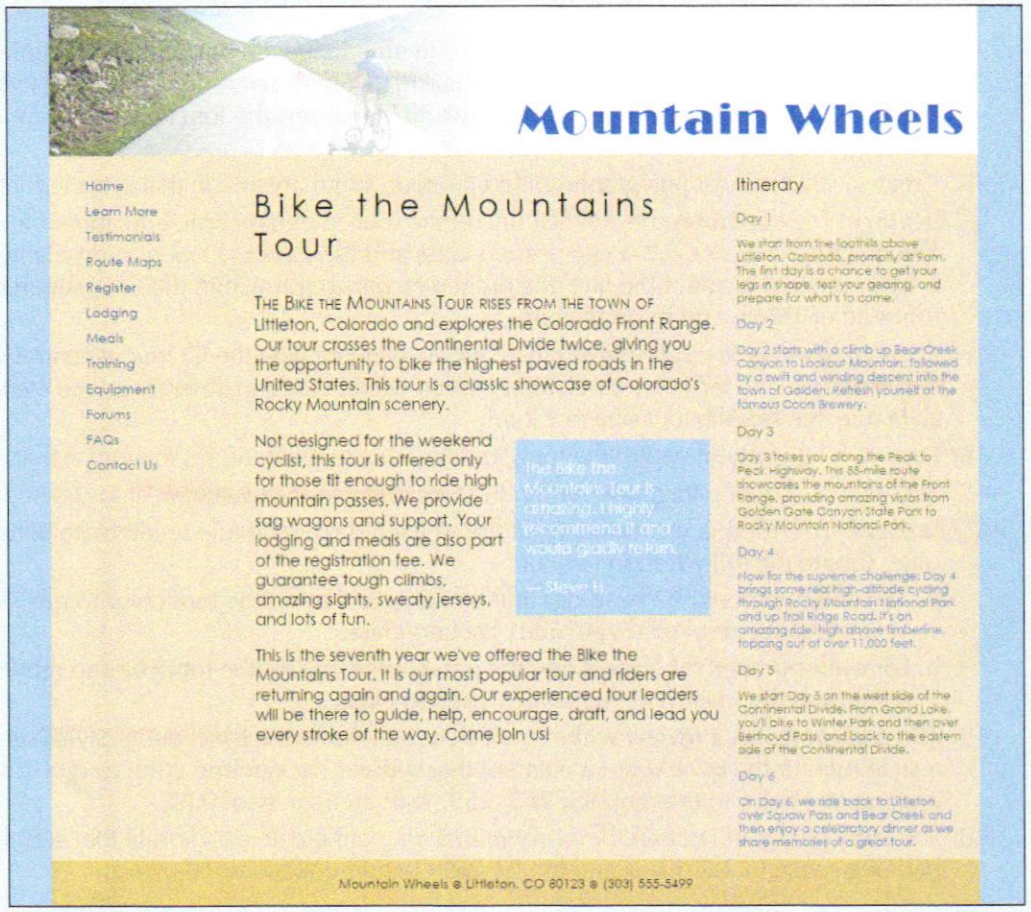
CHALLENGE

Case Problem 2

Data Files needed for this Case Problem: mw_styles_txt.css, mw_tour_txt.html, 1 CSS file, 1 PNG file

Mountain Wheels Adriana and Ivan Turchenko are the co-owners of Mountain Wheels, a bike shop and touring company in Littleton, Colorado. One of their most popular tours is the Bike the Mountains Tour, a six-day excursion over some of the highest roads in Colorado. Adriana wants to update the company's website to provide more information about the tour. She already has had a colleague design a three-column layout with a list of links in the first column and descriptive text in the second and third columns. She has asked for your help in completing the design by formatting the text and colors in the page. Figure 2-55 shows a preview of the design used in the final page.

Figure 2-55 **Description of the Bike the Mountains tour**



Complete the following:

1. Using your editor, open the **mw_tour_txt.html** and **mw_styles_txt.css** files from the html02 ▸ case2 folder. Enter ***your name*** and ***the date*** in the comment section of each file, and save them as **mw_tour.html** and **mw_styles.css** respectively.
2. Go to the **mw_tour.html** file in your HTML editor. Within the document head, create links to the **mw_layout.css** and **mw_styles.css** style sheet files. Study the content and structure of the document and then close the file, saving your changes.
3. Go to the **mw_styles.css** file in your editor. At the top of the file, insert the `@charset` rule to set the encoding for this style sheet to utf-8.
4. Go to the Structural Styles section and create a style rule that sets the background color of the browser window to rgb(173, 189, 227).
5. Create a style rule for the `body` element that sets the background color to rgb(227, 210, 173) and sets the body font to the font stack: 'Century Gothic', sans-serif.
6. Create a style rule to display the body footer with a background color of rgb(208, 184, 109) and set the top and bottom padding space to 5 pixels.
7. Create a style rule for the `address` element to display the text in a normal font with a font size of 0.9em, horizontally center the text, and set the top and bottom padding to 10 pixels.
8. Go to the Heading Styles section and create a style rule to set the font weight of all h1 and h2 headings to normal.
9. Go to the Navigation Styles section and create a style rule for the `nav > ul` selector that removes all list markers, sets the line height to 2em, and sets the font size to 0.9em.
10. For every previously visited or unvisited hypertext link within the navigation list, create a style rule to remove the underlining from the hypertext link and to set the text color to rgb(43, 59, 125).
11. For every hovered or active link within the navigation list, create a style rule to set the text color to rgb(212, 35, 35).
12. Adriana has put information about the tour in an article with the ID "tour_summary". Format this article, starting with the heading. Go to the Article Styles section and create a style rule for `h1` elements nested within the tour_summary article that sets the font size to 2.2em and the letter spacing to 0.2em.
13. Create a style rule for paragraphs within the tour_summary article that sets the font size to 1.1em.
14. **Explore** Adriana wants the first line in the tour_summary article to appear in small capital letters. Use the `first-of-type` pseudo-class and the `first-line` pseudo-element to create a style rule that displays the first line of the first paragraph within the tour_summary article at a font size of 1.2em and in small caps.
15. The tour itinerary is displayed within an `aside` element with the ID *tour_itinerary*. Go to the Aside Styles section and for every `h1` element nested within the tour_itinerary `aside` element, create a style rule that sets the font size to 1.2em.
16. For every h2 element within the tour_itinerary `aside` element, set the font size to 0.9em.
17. Set the font size of paragraphs within the tour_itinerary `aside` element to 0.8em.
18. **Explore** Adriana wants the text color of each day's schedule to alternate between gray and blue. Create the following style rules:
    a. For odd-numbered h2 headings and paragraphs that set the font color to rgb(79, 91, 40). (*Hint*: Use the `nth-of-type(odd)` pseudo-class.)
    b. For even-numbered h2 headings and paragraphs that set the font color to rgb(81, 95, 175). (*Hint*: Use the `nth-of-type(even)` pseudo-class.)
19. The page contains a review within a block quote. Go to the Blockquote Styles section and create a style rule for the `blockquote` element that sets the background color to rgb(173, 189, 227) and the text color to the rgb(255, 255, 255) with an opacity of 0.65.
20. For every paragraph within the `blockquote` element create a style rule that sets the top/bottom padding space to 2.5 pixels and the left/right padding space to 10 pixels.

21. Save your changes to the file and then open the **mw_tour.html** file in your browser. Verify that your design matches that shown in Figure 2-55 including the format applied to the first paragraph of the tour_itinerary article and the alternating colors used in the listing of the itinerary days.

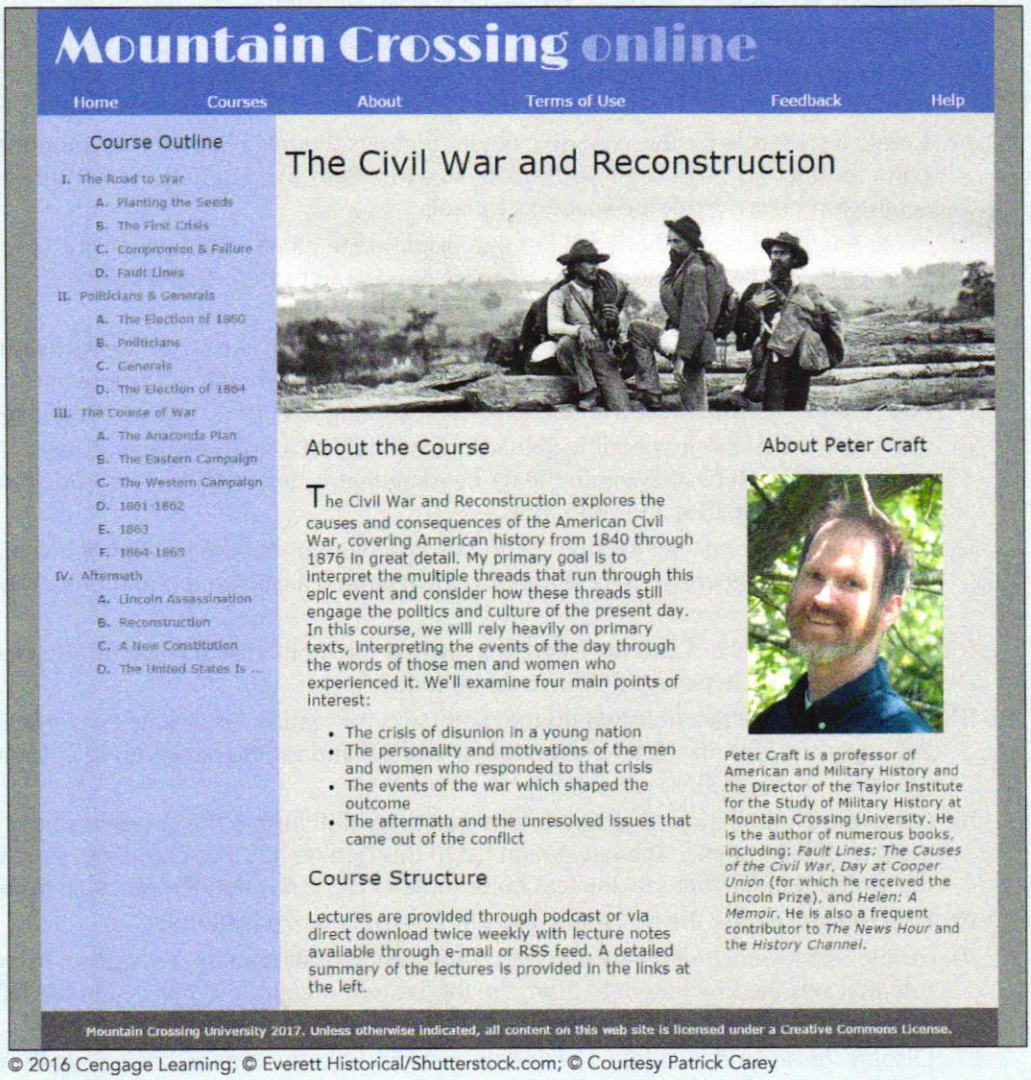
CHALLENGE

Case Problem 3

Data Files needed for this Case Problem: cw_class_txt.html, cw_styles_txt.css, 1 CSS file, 2PNG files

The Civil War and Reconstruction Peter Craft is a professor of military history at Mountain Crossing University. The university is offering a series of online courses, one of which is "The Civil War and Reconstruction" taught by Professor Craft. He has developed the online content and has had a colleague help with the page layout. You've been asked to complete the project by creating text and color styles. A preview of the sample page is shown in Figure 2-56.

Figure 2-56 Civil War History home page



Complete the following:

1. Using your editor, open the **cw_class_txt.html** and **cw_styles_txt.css** files from the html02 ▸ case3 folder. Enter ***your name*** and ***the date*** in the comment section of each file, and save them as **cw_class.html** and **cw_styles.css** respectively.
2. Go to the **cw_class.html** file in your HTML editor. Within the document head, create a link to the **cw_styles.css** style sheet file.
3. **Explore** Using the Google Fonts website, locate the Limelight font. Copy the code for the `link` element to use this font and paste the copied code to the document head in the cw_class.html file.
4. Study the content and structure of the cw_class.html file and then close the file, saving your changes.
5. Go to the **cw_styles.css** file in your editor. At the top of the file, define the character encoding as utf-8.
6. **Explore** On the next line, use the `@import` rule to import the contents of the cw_layout.css file into the style sheet.
7. Go to the Structural Styles section. Within that section create a style rule to set the background color of the browser window to rgb(151, 151, 151).
8. Create a style rule to set the background color of the page body to rgb(180, 180, 223) and set the body text to the font stack: Verdana, Geneva, sans-serif.
9. Display all h1 and h2 headings with normal weight.
10. Create a style rule for every hypertext link nested within a navigation list that removes underlining from the text.
11. Create a style rule for the `footer` element that sets the text color to white and the background color to rgb(101, 101, 101). Set the font size to 0.8em. Horizontally center the footer text, and set the top/bottom padding space to 1 pixel.
12. Next, you'll format the body header that displays the name of the university. Go to the Body Header Styles section and, for the `body > header` selector, create a style rule that sets the background color to rgb(97, 97, 211).
13. The university name is stored in an h1 heading. Create a style rule for the h1 heading that is a direct child of the body header that sets the font size to 4vw with the color value rgba(255, 255, 255, 0.8). Display the text with the font stack: Limelight, cursive. Set the margin space to 0 pixels.
14. The last word of the h1 heading text is enclosed within a `span` element. Create a style rule for the `span` element nested within the h1 heading that is nested within the body header, setting the text color to rgba(255, 255, 255, 0.4).
15. Go the Navigation Styles section. In this section, you format the navigation list that has the ID *mainLinks*. For hypertext links within this navigation list, set the top and bottom padding space to 5 pixels.
16. For previously visited and unvisited links within the mainLinks navigation list, create a style rule that displays the hypertext links in a white font.
17. For hovered or active links within the mainLinks navigation list, create a style rule that displays the hypertext links in white with an opacity of 0.8 and set the background color to the value rgba(51, 51, 51, 0.5).
18. Go to the Outline Styles section. In this section, you'll format the course outline that appears on the page's left column. The navigation list in this outline has the ID *outline*. Create a style rule for this navigation list that sets the text color to rgb(51, 51, 51) and the font size to 0.8em.
19. Horizontally center the h1 headings within the outline navigation list.
20. For the first level `ol` elements that are a direct child of the outline navigation list, create a style rule that sets the line height to 2em, the top/bottom margin to 0 pixels and the left/right margin to 5 pixels. Display the list marker as an upper-case Roman numeral.
21. Display the second level of `ol` elements nested within the outline navigation list with an upper-case letter as the list marker.
22. Display all previously visited and unvisited links in the outline navigation list using the color value rgb(101, 101, 101).
23. Display hovered and active links in the outline navigation list using the color value rgb(97, 97, 211) with the text underlined.

24. Go to the Section Styles section. In this section, format the description of the course. Create a style rule that sets the background color of the `section` element to rgb(220, 220, 220).
25. Format the heading of this section by creating a style rule for the `section header h1` selector that sets the font size of 2.2em and the left padding space to 10 pixels.
26. Go to the Article Styles section and create a style rule for h2 headings within the `article` element that sets the font size to 1.4em.
27. **Explore** Display the first letter of the first paragraph within the `article` element with a font size of 2em and vertically aligned with the baseline of the surrounding text. (*Hint*: Use the `first-of-type` pseudo-class and the `first-letter` pseudo-element.)
28. Information about Peter Craft has been placed in an `aside` element. Go to the Aside Styles section and create a style rule that sets the font size of text in the `aside` element to 0.9em.
29. For h1 headings nested within the `aside` element, create a style rule that sets the font size to 1.4em and horizontally centers the text.
30. Save your changes to the file and then open the **`cw_class.html`** file in your browser. Verify that the appearance of the page resembles that shown in Figure 2-56. Confirm that when you change the width of the browser window, the size of the page heading text changes in response to setting the heading text using the vw unit.

CREATE

Case Problem 4

Data Files needed for this Case Problem: lake_home_txt.html, lake_styles_txt.css, 1 CSS file, 2 PNG files, 2 TTF files, 2 WOFF files

The Great Lakescape Lodge Ron Nelson is the owner of The Great Lakescape Lodge in Baileys Harbor, Wisconsin. He has hired you to work on the redesign of the lodge's website. You'll start by working on the site's home page. Ron has already written the text of the page, gathered all of the graphic files, and had a colleague design the page layout. He wants you to work on the page's color scheme and typography. A possible solution is shown in Figure 2-57.

Figure 2-57 **Home page of the Great Lakescape Lodge**

Complete the following:

1. Using your editor, open the **lake_home_txt.html** and **lake_styles_txt.css** files from the html02 ▸ case4 folder. Save them as **lake_home.html** and **lake_styles.css** respectively.
2. Go to the **lake_home.html** file in your editor and link it to the **lake_layout.css** and **lake_styles.css** style sheet file. Take some time to study the content and structure of the document and then save your changes to the file.
3. Go to the **lake_styles.css** file in your editor and begin creating the color scheme and typographic styles for the lodge's home page. The final design is up to you but it should include the following features:
   - Definition of the character encoding used in the style sheet file
   - Application of a web font (Two fonts are supplied for you in the html02 ▸ case4 folder.)
   - Setting background and text colors using both color values and color names
   - An application of a semi-transparent color
   - Selectors showing style rules applied to nested elements, child elements, and elements based on the `id` attribute
   - Styles that modify the appearance of list and list markers
   - Use of pseudo-elements and pseudo-classes as selectors
   - Styles that modify the padding space and margin space around an element
   - A style rule to generate content in the rendered page
4. Include informative style comments throughout the style sheet
5. Save your completed style sheet.