4

How to use CSS to format the elements of a web page

After you code the HTML that defines the structure of a web page, you're ready to code the CSS style rules that determine how the page is formatted. To do that, you need to learn how to code selectors, and you need to learn how to code the properties and values for style rules.

In this chapter, you'll learn how to code all types of selectors, and you'll learn how to apply the CSS properties for formatting text. Then, in the next chapter, you'll learn how to use the CSS box model for doing other types of formatting.

An introduction to CSS

Before you code the CSS for a web page, you need to know how to provide the CSS for a web page. You also need to know how to make the HTML5 semantic elements work in older browsers and how to make sure that the elements of a page are rendered the same in every browser.

Three ways to provide CSS styles for a web page

Figure 4-1 shows three ways to provide CSS styles for a web page. First, you can code a link element that refers to an *external style sheet*. That's a separate file that contains the CSS for the page. This separates the content from the formatting and makes it easy to use the same styles for more than one page.

The attributes for a link element that links to an external file are the rel (relationship) attribute with a value of "stylesheet", and the href attribute that locates the file. As you learned in the last chapter, href attributes are usually coded with a URL that is relative to the current file. As a result, the relative URL in the first example goes down one folder to the styles folder and locates a file named main.css.

Second, you can embed a CSS style sheet in the HTML for a page. This is referred to as an *embedded style sheet*. When you embed a style sheet, the CSS style rules are coded in a style element in the head section of the HTML. For instance, the embedded style sheet in the first group contains one style rule for the body element and another for the h1 element. This works okay if the styles are only going to be used for that one document, but otherwise it's better to use an external style sheet.

Third, you can use *inline styles* within an HTML document as shown by the third example. When you use an inline style, you code a style attribute for the HTML element with a value that contains all the CSS declarations that apply to the element. For instance, the inline style in this example applies two CSS declarations to the h1 element. Unfortunately, this type of formatting means that the content and formatting are tightly linked, so this can quickly get out of control.

If you use more than one way to provide styles for a page, the styles that are applied last override the styles that are applied earlier. This is illustrated by the example in this figure. Here, an inline style sets the font size for h1 elements, and that will override the embedded style that sets the font size for h1 elements.

The next example in this figure shows that you can include more than one style sheet in a single document. Then, the styles are applied from the first external style sheet to the last. Here again, a style in the last style sheet will override a style for the same element in an earlier style sheet.

Three ways to provide styles

Use an external style sheet by coding a link element in the head section

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

Embed the styles in the head section

```
<style>
    body {
        font-family: Arial, Helvetica, sans-serif;
        font-size: 100%; }
    h1 { font-size: 250%; }
</style>
```

Use the style attribute to apply styles to a single element

```
<h1 style="font-size: 500%; color: red;">Valley Town Hall</h1>
```

The sequence in which styles are applied

- Styles from an external style sheet
- Embedded styles
- Inline styles

A head element that includes two style sheets

```
<head>
    <title>San Joaquin Valley Town Hall</title>
    <link rel="stylesheet" href="../styles/main.css">
    <link rel="stylesheet" href="../styles/speaker.css">
</head>
```

The sequence in which styles are applied

- From the first external style sheet to the last

Description

- When you use *external style sheets*, you separate content (HTML) from formatting (CSS). That makes it easy to use the same styles for two or more documents.
- If you use *embedded styles*, you have to copy the styles to other documents before you can use them a second time.
- If you use *inline styles* to apply styles, the formatting is likely to get out of control.
- If more than one declaration for the same property is applied to the same element, the last declaration overrides the earlier declarations.
- When you specify a relative URL for an external CSS file, the URL is relative to the current file.

Figure 4-1 Three ways to provide CSS styles for a web page

Two ways to provide for browser compatibility

To make sure that the HTML5 semantic elements will work in older browsers like Internet Explorer 7 and 8, web developers have been using a JavaScript file known as a *shiv* (or *shim*). Then, when the page is loaded, the shiv is loaded and executed. That forces an old browser to recognize the HTML5 semantic elements, and it adds a CSS style rule to the page that tells older browsers that these elements are block elements, not inline elements.

To run this shiv for a web page, you simply add the script element shown at the top of figure 4-2 to the head element of each HTML document that uses HTML5 elements. Here, the src attribute provides the URL that accesses the shiv.

Today, however, the only browsers that don't already support the HTML5 semantic elements are the versions of Internet Explorer before version 9. So, unless you know that your users will be using these older browsers, you no longer need to use the JavaScript shiv. That's why none of the examples in this book use the shiv.

The other problem that faces web developers today is that the five modern browsers render some elements of a web page differently. One example is the inline abbr element that's used to define abbreviations.

In Firefox and Chrome, the textual content for this element is displayed with a dotted line below it. Then, when the user points to the element with the mouse, a tooltip that defines the abbreviation is displayed. In contrast, a dotted line isn't displayed under the abbreviation in Internet Explorer or Edge. Because of that, the user won't know that there is an abbreviation on the page unless he happens to point to it. Then, a tooltip appears just as it does in Firefox.

To standardize how elements like this are displayed, you can use the normalize.css style sheet that's described in this figure. This style sheet applies styles to elements like the abbr element so they appear exactly the same in all browsers. This can save you a lot of time dealing with small rendering issues near the end of a web development project.

The normalize.css style sheet also sets the margins for the body of the document to zero. That means that there's no space between the body and the edge of the browser window. This is important because different browsers provide for different margins for the body. You'll learn more about working with the margins for the body as well as other elements in chapter 5.

A third example of what the normalize.css style sheet does is that it sets the default font family for the document to sans-serif. You'll learn more about font families later in this chapter. When you do, you'll learn that sans-serif fonts are easiest to read in a browser. Even so, the default font for all of the major browsers is a serif font. If you use the normalize.css style sheet, then, you only need to set the font family if you want to use a specific sans-serif font.

For reasons like these, many web developers use the normalize.css style sheet for all of their pages. That's also why this style sheet is used in all of the web applications for this book.

The JavaScript shiv for using HTML5 semantics with IE 7 and 8

```
<head>
<script
  src="https://cdnjs.cloudflare.com/ajax/libs/html5shiv/3.7.3/html5shiv.js">
</script>
</head>
```

What the shiv does

- A JavaScript *shiv* (or *shim*) forces older browsers, like Internet Explorer 7 and 8, to recognize the HTML5 semantic elements and let you apply CSS to these elements.
- The effect of the shiv is to add the HTML5 semantic elements to the DOM in the browser and also to add a CSS style rule that tells the browser to treat the HTML5 elements as block elements instead of inline elements.
- Because the use of the older IE browsers accounts for a small percentage of browser usage, none of the examples in this book use the shiv.

The URL for downloading the normalize.css style sheet

- http://necolas.github.io/normalize.css/

How to download normalize.css and save it to your website

- Open a browser, browse to the URL shown above, and click the Download button.
- If CSS files are associated with a program that's installed on your computer, the normalize.css file will be opened in that program. Then, you can use that program to save the file to your website.
- If CSS files aren't associated with a program on your computer, the normalize.css file will be displayed on a page within the browser. Then, you can right-click the page, choose Save As, and use the dialog box that's displayed to save the normalize.css file to your website.
- Once you save the normalize.css file to your website, you can code a link element for it in each page. This element must be coded before the link elements for other style sheets.

What the normalize.css style sheet does

- Normalize.css is a style sheet that makes minor adjustments to browser defaults so all browsers render HTML elements the same way.
- For instance, the normalize.css style sheet sets the margins for the body of the document to zero so there's no space between the body and the edge of the browser window. See chapter 5 for more information on margins.
- The normalize.css style sheet also sets the default font family for the document to sans-serif. See figure 4-11 in this chapter for more information on working with font families.
- Because the normalize.css style sheet resolves minor browser variations, all of the web pages from this point on in this book will use it.

Figure 4-2 Two ways to provide for browser compatibility

How to specify measurements and colors

For many of the properties of a style rule, you will need to know how to specify measurements and colors. So let's start there.

How to specify measurements

Figure 4-3 shows the five units of measure that are commonly used with CSS: pixels, points, ems, rems, and percents. Here, the first two are *absolute units of measure*, and the next three are *relative units of measure*. Other absolute units are inches and picas, but they aren't used as often.

When you use relative units of measure like ems, rems, or a percent, the measurement will change if the user changes the browser's font size. If, for example, you set the size of a font to 80 percent of the browser's default font size, that element will change if the user changes the font size in the browser. Because this lets the users adjust the font sizes to their own preferences, we recommend that you use relative measurements for font sizes.

In contrast, when you use an absolute unit of measure like pixels or points, the measurement won't change even if the user changes the font size in the browser. If, for example, you set the width of an element in pixels and the font size in points, the width and font size won't change.

When you use pixels, though, the size will change if the screen resolution changes. That's because the screen resolution determines the number of pixels that are displayed on the monitor. For instance, the pixels on a monitor with a screen resolution of 1280 x 1024 are closer together than the pixels on the same monitor with a screen resolution of 1152 x 864. That means that a measurement of 10 pixels will be smaller on the screen with the higher resolution. In contrast, a point is 1/72nd of an inch no matter what the screen resolution is.

The examples in this figure show how you can use pixels, percentages, and ems in your CSS. Here, the bottom border for the header is set to 3 pixels. In contrast, the font sizes are set as percentages, and the margins and padding are set as ems. This is a typical way to use these measurements.

To start, a font size of 100% is applied to the body element. This means that the font will be set to 100% of the default font size for the browser, which is usually 16 pixels. Although you can get the same result by omitting the font-size property, this property is typically included to make it clear that the default font size for the browser will be used.

Left and right margins of 2 ems are also applied to the body element. Because an em is equal to the current font size, the body element is indented 32 pixels (2 times 16 pixels). These properties are included because the normalize.css style sheet that's used with this page sets these margins to zero. Note, however, that it's not necessary to set the top and bottom margins for the body. That's because the h1 element has a default top margin and the <p> element has a default bottom margin.

Common units of measure

Symbol	Name	Type	Description
px	pixels	absolute	A pixel represents a single dot on a monitor. The number of dots per inch depends on the resolution of the monitor.
pt	points	absolute	A point is 1/72 of an inch.
em	ems	relative	One em is equal to the font size for the current font.
rem	rems	relative	One rem is equal to the font size for the root element.
%	percent	relative	A percent specifies a value relative to the current value.

The HTML for a web page

```
<body>
    <header>
        <h1>San Joaquin Valley Town Hall</h1>
    </header>
    <main>
        <p>Welcome to San Joaquin Valley Town Hall. We have some
           fascinating speakers for you this season!</p>
    </main>
</body>
```

CSS that uses relative units of measure with a fixed border

```
body {
    font-size: 100%;
    margin-left: 2em;
    margin-right: 2em; }
header {
    padding-bottom: .75em;
    border-bottom: 3px solid black;
    margin-bottom: 0; }
h1 {
    font-size: 200%;
    margin-bottom: 0; }
```

The web page in a web browser

Description

- You use the units of measure to specify a variety of CSS properties, including font-size, line-height, width, height, margin, and padding.
- To specify an *absolute measurement*, you can use pixels or points.
- To specify a *relative measurement*, you can use ems, rems, or percents. This type of measurement is relative to the size of another element.

Figure 4-3 How to specify measurements

Next, the padding below the header element is set to .75 em, or 12 pixels (16 times .75). Then, a solid black border that's 3 pixels wide is added below the padding, and the margin below the border is set to 0. For the margin bottom, no unit of measure is specified because it doesn't matter what the unit of measure is when the value is zero.

The last style rule indicates that the h1 element should be 200% of the current font size, which was set in the style rule for the body element. Also, the margin below the heading should be zero.

When you use ems or percentages to specify font sizes for two or more levels of child elements, it can sometimes be difficult to keep track of the current font size. In that case, you might want to consider using rems instead of ems. Unlike ems, which are relative to the current font size, rems are relative to the size of the root, or html, element. Then, if you specify a font size for the html element, you can base all other font sizes on that size. You can also let the font size for the html element default to the browser's font size and base all other font sizes on that size. Note that we don't use rems in this book because our pages are structured so it's easy to determine the current font size.

How to specify colors

Figure 4-4 shows three ways to specify colors. The easiest way is to specify a color name, and this figure lists the names for 16 colors that are supported by all browsers. In addition to these names, though, most browsers support the color names in the CSS3 Color specification. To find a complete list of these color names, you can go to the website listed in the next figure.

Another way to specify a color is to use an *RGB* (red, green, blue) *value*. One way to do that is to specify the percent of red, green, and blue that make up the color. For instance, the example in this figure specifies 100% red, 40% green, and 20% blue. When you use this method, you can also use any values from 0 through 255 instead of percents. Then, 0 is equivalent to 0% and 255 is equivalent to 100%. This gives you more precision over the resulting colors.

The third way to specify a color is to use *hexadecimal*, or *hex*, *values* for the red, green, and blue values, and this is the method that has been preferred by most web designers. In hex, a value of "000000" results in black, and a value of "FFFFFF" results in white. The simple conversion of percentages to hex is 0% is 00, 20% is 33, 40% is 66, 60% is 99, 80% is CC, and 100% is FF. When you use this technique, the entire value must be preceded by the hash character (#).

When you use complex values for colors, you usually get the hex value that you want from a chart or palette that shows all of the colors along with their hex values. For instance, you can get a complete list of the hex values for colors by going to the website listed in this figure. Or, if you're using an IDE like Dreamweaver CC, you may also be able to choose a color from a palette and then have the IDE insert the hex value for that color into your code. A third alternative is to get the right hex values from printed color charts, which are commonly found in graphics design books.

16 descriptive color names

black	silver	white	aqua	gray	fuchsia
red	lime	green	maroon	blue	navy
yellow	olive	purple	teal		

Three ways to specify colors

With a color name

```
color: silver;
```

With an RGB (red-green-blue) value

```
color: rgb(100%, 40%, 20%);
color: rgb(255, 102, 51);   /* Using multiples of 51 from 0 to 255 */
```

With an RGB value that uses hexadecimal numbers

```
color: #ffffff;             /* This color is white */
color: #000000;             /* This color is black */
color: #ff0000;             /* This color is red */
```

CSS that uses hexadecimal values to specify colors

```
body {
    font-size: 100%;
    margin-left: 2em;
    background-color: #FFFFCC; }   /* This could also be coded as #FFC */
h1 {
    font-size: 200%;
    color: #00F; }                 /* This could also be coded as #0000FF */
```

The HTML in a web browser

Accessibility guideline

- Remember the visually-impaired. Dark text on a light background is easier to read, and black type on a white background is easiest to read.

Description

- All browsers support the 16 color names shown above, and most browsers support many more. These are okay for getting started with the use of colors.
- Most graphic designers use *hexadecimal*, or *hex*, *values* to specify an *RGB value* because that lets them choose from over 16 million colors. You can find a list of color names and their corresponding hex values at http://www.w3.org/TR/css3-color.
- With IDEs like Dreamweaver CC, you can select a color from a palette of colors and let the IDE insert the right color codes into your style rules in either RGB or hex format.

Figure 4-4 How to specify colors

Before I go on, you should realize that the color property determines the foreground color, which is the color of the text. In the CSS in this figure, for example, the color of the h1 element is set to blue (#00F or #0000FF), and the background color of the body is set to a light yellow (#FFFFCC or #FFC). You'll learn more about setting background colors in the next chapter.

You should also realize that the color property for an element is *inherited* by any child elements. If, for example, you set the color property of the body element to navy, that color will be inherited by all of the elements in the body of the document. However, you can override an inherited property by coding a style rule with a different value for that property. Throughout this chapter and book, I'll point out the properties that are inherited as well as those that aren't.

Whenever you're using colors, please keep the visually-impaired in mind. For anyone, dark text on a light background is easier to read, and black on white is easiest to read. In general, if your pages are hard to read when they're displayed or printed in black and white, they aren't good for the visually-impaired. On that basis, the example in this figure could be improved.

How to use the CSS3 color specifications

To provide even more color options for web designers, CSS3 lets you code color specifications in three more ways. These are summarized in figure 4-5.

First, you can use *RGBA values*. This works like RGB values, but with a fourth parameter that provides an opacity value. If, for example, you set this value to 0, the color is fully transparent so anything behind it will show through. Or, if you sent this value to 1, nothing will show through.

Second, you can use *HSL values*. To do that, you provide a number from 1 through 359 that represents the hue that you want. The hue is one of the main properties of a color. Then, you can provide a number from 0 through 100 that represents the saturation percent with 100 being the full hue. Last, you can provide a number from 0 through 100 that represents the lightness percent with 50 being normal, 100 being white, and 0 being black.

Third, you can use *HSLA values*. This is just like HSL values, but with a fourth parameter that provides an opacity value between 0 and 1.

The examples in this figure give you some idea of how these values work, especially if you see them in color in the eBook or in the examples you can download for this book. Note here that the second and third examples are the same hue, but the saturation and lightness percents make them look quite different. This should give you some idea of the many color variations that CSS3 offers. But here again, please keep accessibility in mind whenever you're using colors.

Besides providing for RGBA, HSL, and HSLA values, CSS3 also provides 147 more keywords for colors that are generally supported by modern browsers. For a complete listing, you can go to the URL in this figure.

Three ways to code CSS3 colors

The syntax for RGBA colors

```
rgba(red%, green%, blue%, opacity-value)
```

The syntax for HSL and HSLA colors

```
hsl(hue-degrees, saturation%, lightness%)
hsla(hue-degrees, saturation%, lightness%, opacity-value)
```

Values	Description
`opacity-value`	A number from 0 to 1 with 0 being fully transparent and 1 being fully opaque.
`hue-degrees`	A number of degrees ranging from 0 to 359 that represents the color.
`saturation%`	A percentage from 0 to 100 with 0 causing the hue to be ignored and 100 being the full hue.
`lightness%`	A percentage from 0 to 100 with 50 being normal lightness, 0 being black, and 100 being white.

Examples

```
h1 { color: rgba(0, 0, 255, .2)        /* transparent blue */ }
h1 { color: hsl(120, 100%, 25%)        /* dark green */ }
h1 { color: hsl(120, 75%, 75%)         /* pastel green */ }
h1 { color: hsla(240, 100%, 50%, 0.5)  /* semi-transparent solid blue */ }
```

The colors in a browser

Description

- *RGBA* enhances the RGB specification by providing a fourth value for opacity.
- With *HSL* (Hue, Saturation, and Lightness) and *HSLA*, you specify the number of hue degrees for a color. Then, you can enhance the hue by providing for both saturation and lightness percentages. HLSA also offers a fourth value for opacity.
- CSS3 also provides 147 more keywords for colors that are generally supported by modern browsers. For a complete listing, go to:
  `http://www.w3.org/TR/SVG/types.html#ColorKeywords`
- With an IDE like Dreamweaver, you can select the type of color specification you want to use (RGBA, HSL, or HSLA). Then, you can select the color or hue, saturation, lightness, and opacity.

Figure 4-5 How to use the CSS3 color specifications

How to code selectors

Now, you're ready to learn how to code selectors. Once you understand that, you will be able to apply CSS formatting to any elements in a web page.

How to code selectors for all elements, element types, ids, and classes

Figure 4-6 presents the four types of selectors that you'll use the most. To start, this figure shows the body of an HTML document that contains a main and a footer element. Here, the two <p> elements in the main element have class attributes with the value "blue". Also, the <p> element in the footer has been assigned an id of "copyright", and it has a class attribute with two values: "blue" and "right". This means that this element is assigned to two classes.

Next, this figure shows the CSS style rules that are used to format the HTML. Here, the style rule in the first example uses the universal selector (*) so it applies to all HTML elements. This sets the top and bottom margins for all elements to .5em and the left and right margins to 1em. The exception is if the normalize.css style sheet is linked to the document. Then, the margins specified by the universal selector don't override the margins that are set for any specific elements in normalize.css. Because of that, you probably won't use the universal selector often.

The three style rules in the second group of examples select elements by type. These are referred to as type selectors. To code a type selector, you just code the name of the element. As a result, the first style rule in this group selects the main element, the second style rule selects all h1 elements, and the third style rule selects all p elements. These style rules set the border and padding for the main element, the font family for the h1 elements, and the left margin for all <p> elements.

The style rule in the third group of examples selects an element by its id. To do that, the selector is a hash character (#) followed by an id value that uniquely identifies an element. As a result, this style rule selects the <p> element that has an id of "copyright". Then, its declaration sets the font size for the paragraph to 80% of the font size for the page.

The two style rules in the last group of examples select HTML elements by class. To do that, the selector is a period (.) followed by the class name. As a result, the first style rule selects all elements that have been assigned to the "blue" class, which are all three <p> elements. The second style rule selects any elements that have been assigned to the "right" class. That is the paragraph in the footer. Then, the first style rule sets the color of the font to blue and the second style rule aligns the paragraph on the right.

One of the key points here is that a class attribute can have the same value for more than one element on a page. Then, if you code a selector for that class, it will be used to format all the elements in that class. In contrast, since the id for an element must be unique, an id selector can only be used to format a single element.

HTML that can be selected by element type, id, or class

```
<main>
    <h1>This Season's Speaker Lineup</h1>
    <p class="blue">October: Jeffrey Toobin</p>
    <p class="blue">November: Andrew Ross Sorkin</p>
</main>
<footer>
    <p id="copyright" class="blue right">Copyright 2018</p>
</footer>
```

CSS style rules that select by element type, id, and class

All elements

```
* { margin: .5em 1em; }
```

Elements by type

```
main {
    border: 2px solid black;
    padding: 1em; }
h1 { font-family: Arial, sans-serif; }
p { margin-left: 3em; }
```

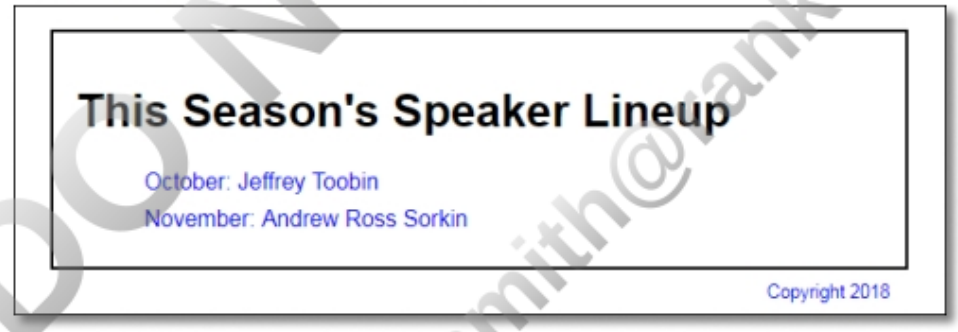
One element by ID

```
#copyright { font-size: 80%; }
```

Elements by class

```
.blue { color: blue; }
.right { text-align: right; }
```

The elements displayed in a browser



Description

- You code a selector for all elements by coding the *universal selector* (*).
- You code a selector for all elements of a specific type by naming the element. This is referred to as a *type selector*.
- You code a selector for an element with an id attribute by coding a hash character (#) followed by the id value.
- You code a selector for an element with a class attribute by coding a period followed by the class name. Then, the style rule applies to all elements with that class name.

Figure 4-6 How to code selectors for all elements, element types, ids, and classes

Incidentally, the margin-left property for the <p> elements overrides the margin setting for all elements, which of course includes all <p> elements. That's why the two paragraphs in the main element are indented.

How to code relational selectors

Figure 4-7 shows how to code *relational selectors*. As you read about these selectors, keep in mind that terms like *parent*, *child*, *sibling*, and *descendent* are used in the same way that they are in a family tree. Child elements are at the first level below a parent element. Sibling elements are at the same level. And descendent elements can be one or more levels below a parent element.

That means a *descendant selector* selects all the elements that are contained within another element. For instance, all of the elements in the HTML in this figure are descendents of the main element, the li elements are also descendents of the ul element, and the <a> elements are descendents of the li, ul, and main elements.

To code a descendant selector, you code a selector for the parent element, followed by a space and a selector for the descendent element. This is illustrated by the first group of examples. Here, the first selector selects all li elements within the main element. Then, the second descendant selector selects all the <a> elements that are descendants of the ul element.

The next group of examples shows how to use an *adjacent sibling selector* to select an element that's coded at the same level as another element and is also coded right next to it. For instance, the h1, ul, h2, and <p> elements in the HTML are all siblings, and the h1 and ul elements are adjacent siblings. In contrast, the h1 element and the <p> elements aren't adjacent siblings, but the h2 element and the first <p> element are adjacent siblings.

To code an adjacent sibling selector, you code a selector for the first element, followed by a plus sign and a selector for the sibling element. In the example in this group, the selector will select any <p> elements that are adjacent to h2 elements. That means it will select the first <p> element that follows the h2 element.

If you want to select elements only when they're child elements of a parent element, you can code a *child selector*. To do that, you separate the parent and child selector with a greater than (>) sign. In this figure, for example, the first child selector selects the <p> elements that are children of the main element, and the second selector selects the <a> elements that are children of the li elements.

Unlike the adjacent sibling selector, a *general sibling selector* selects any sibling element whether or not the elements are adjacent. To code this type of selector, you separate the selector for the first element and the selector for the sibling element by a tilde (~). In this figure, the general sibling selector selects all the <p> elements that follow any h2 element.

HTML that can be selected by relationships

```
<main>
    <h1>This Season's Town Hall speakers</h1>
    <ul class="speakers">
        <li>January: <a href="speakers/brancaccio.html">
            David Brancaccio</a></li>
        <li>February: <a href="speakers/fitzpatrick.html">
            Robert Fitzpatrick</a></li>
        <li>March: <a href="speakers/williams.html">
            Juan Williams</a></li>
    </ul>
    <h2>Post-lecture luncheons</h2>
    <p>Extend the excitement by going to the luncheon</p>
    <p>A limited number of tickets are available.</p>
    <p><em>Contact us by phone</em> at (559) 555-1212.</p>
</main>
```

CSS style rules with relational selectors

Descendant

```
main li { font-size: 90%; }
ul a { color: green; }
```

Adjacent sibling

```
h2+p { margin-top: .5em; }
```

Child

```
main>p { font-size: 80%; }
li>a { color: green; }
```

General sibling (CSS3)

```
h2~p { margin-left: 2em; }
```

Description

- When you use relational selectors, you can often avoid the need for id or class attributes.
- To select elements only when they are descendants of a higher-level element, use a *descendant selector* that consists of the higher element, a space, and the descendent element.
- To select a sibling element that's adjacent to another element, use an *adjacent sibling selector* that consists of the first element, a plus sign (+), and the sibling element.
- To select elements only when they are child elements of the parent element, you can use a *child selector* that consists of the parent element, the greater than sign (>), and the child element.
- To select any elements that are siblings to another element, you can use a *general sibling selector* that consists of the first element, a tilde (~), and the sibling element, but this can only be used by browsers that support CSS3.

Figure 4-7 How to code relational selectors

How to code combinations of selectors

The first group of examples in figure 4-8 shows how to code selector combinations. To select an element type by class name, for example, you code the element name, followed by a period and the class name. Here, the first style rule selects ul elements that have a class of "speakers".

You can also code multiple selectors for the same style rule. To do that, you separate the selectors with commas as shown in the next group of examples. Here, the first style rule uses multiple selectors to apply its declarations to all h1, h2, and h3 elements. Then, the second style rule uses multiple selectors to apply its declarations to all <p> elements and to li elements that are descendents of the ul elements that are assigned to the speakers class.

How to code attribute selectors

An *attribute selector* selects elements based on an attribute or attribute value. This is illustrated by the second group of examples in figure 4-8. Although you may never need to code attribute selectors for your HTML code, these selectors are often used by JavaScript.

To code an attribute selector that selects elements that have an attribute, you code an element selector followed by the name of the attribute in brackets [ ]. For instance, the first attribute example uses the universal selector to select all elements that have an href attribute. Note, however, that you can also omit the universal selector and code just the attribute in brackets to select all elements with that attribute. In contrast, the second attribute selector selects all of the <a> elements that have an href attribute.

To code an attribute selector that selects elements with the specified attribute value, you follow the attribute name with an equals sign and the value enclosed in quotation marks. This is illustrated in the third example. This selector will select all input elements with a type attribute whose value is set to "submit".

Combinations of selectors

A selector for a class within an element

```
ul.speakers { list-style-type: square; }
```

Multiple selectors

```
h1, h2, h3 { color: blue; }
p, ul.speakers li { font-family: "Times New Roman", serif; }
```

Attribute selectors

All elements with href attributes

```
*[href] { font-size: 95%; }
```

All <a> elements with href attributes

```
a[href] { font-family: Arial, sans-serif; }
```

All input elements with type attributes that have a value of "submit"

```
input[type="submit"] {
    border: 1px solid black;
    color: #ef9c00;
    background-color: #facd8a; }
```

Description

- To code a selector for an element and class, code the element name, a period, and the class name.
- To code multiple selectors for the same style rule, use commas to separate the selectors.
- To select all elements with a specific attribute, you can use an *attribute selector* that consists of the universal selector followed by the attribute name in brackets. You can also omit the universal selector when you code this type of selector.
- To select elements with a specific attribute, you can use an attribute selector that consists of the element followed by the attribute name within brackets.
- To select an element with a specific attribute value, you can use an attribute selector that consists of the element followed by the attribute name, an equal sign, and a value within quotation marks.
- When you're coding the CSS for an HTML page, you usually don't need attribute selectors. They're more useful when you're using JavaScript.

Figure 4-8 How to code combinations and attribute selectors

How to code pseudo-class and pseudo-element selectors

Figure 4-9 shows how to code *pseudo-class* and *pseudo-element* selectors. To code pseudo-class selectors, you use the classes in the first two tables of this figure. These classes represent conditions that apply to the elements on a page. For example, you can use the :link pseudo-class to refer to a link that hasn't been visited, the :hover pseudo-class to refer to the element that has the mouse hovering over it, and the :focus pseudo-class to refer to the element that has the focus.

You can also use the CSS3 pseudo-classes to refer to specific relationships. For example, the :first-child class refers to the first child of an element, and the :only-child class refers to the only child for an element that has only one. Although some older browsers don't support these classes, that's okay if these selectors are only used for graphics.

The third table in this figure presents CSS3 pseudo-elements that you can use to select portions of text. For instance, you can use the ::first-line element to refer to the first line in a paragraph that consists of more than one line. Although these pseudo-elements were available in earlier versions of CSS, they were coded with a single colon just like the pseudo-classes. With CSS3, the pseudo-elements were redefined with double colons to help distinguish them from the pseudo-classes. Because of that, it's best to code the pseudo-elements with two colons as shown in the fourth example in this figure. The exception is if the formatting needs to work in older browsers like versions of IE before IE9.

In the examples, the first pseudo-class selector causes all links to be displayed in green. Then, the second selector is a combination selector that applies to any link that has the mouse hovering over it or the focus on it. As the accessibility guideline in this figure indicates, you should always code the hover and focus pseudo-classes for links in combination so the formatting is the same whether the user hovers the mouse over a link or tabs to it.

The third example uses a pseudo-class selector that causes the text in the first <p> element in the main element to be boldfaced. You can see how this works in the browser display. Note here that the second <p> element isn't boldfaced because it isn't the first child in the main element.

The fourth example takes this one step further by combining a pseudo-class and a pseudo-element in a selector. As a result, the first letter of the first child in the main element is larger than the other letters in the paragraph.

Although the pseudo-class and pseudo-element selectors in this figure are the ones you'll use most often, there are others that can be useful. In chapter 7, for example, you'll learn how to use the ::after pseudo-element to format a multi-tier navigation menu. And in chapter 12, you'll be introduced to some CSS3 pseudo-classes that apply to tables. For a complete list of these classes and elements, you can go to the W3C documentation on the web.

Common CSS pseudo-classes

`:link`	A link that hasn't been visited. By default, blue, underlined text.
`:visited`	A link that has been visited. By default, purple, underlined text.
`:active`	The active link (mouse button down but not released). By default, red, underlined text.
`:hover`	An element with the mouse hovering over it. Code this after :link and :visited.
`:focus`	An element like a link or form control that has the focus.

Common CSS3 pseudo-classes

`:first-child`	The first child of an element.
`:last-child`	The last child of an element.
`:only-child`	The only child of an element.

Common CSS3 pseudo-elements

`::first-letter`	The first letter of an element.
`::first-line`	The first line of an element.

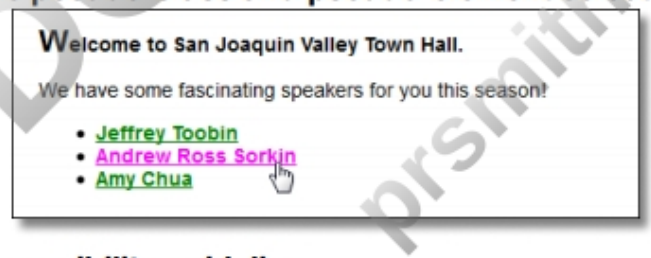
HTML that can be used by pseudo-class and pseudo-element selectors

```
<main>
    <p>Welcome to San Joaquin Valley Town Hall.</p>
    <p>We have some fascinating speakers for you this season!</p>
    <ul>
        <li><a href="toobin.html">Jeffrey Toobin</a></li>
        <li><a href="sorkin.html">Andrew Ross Sorkin</a></li>
        <li><a href="chua.html">Amy Chua</a></li></ul>
</main>
```

The CSS for pseudo-class and pseudo-element selectors

```
a:link { color: green; }
a:hover, a:focus { color: fuchsia }
main p:first-child { font-weight: bold; }
main p:first-child::first-letter { font-size: 150% }
```

The pseudo-class and pseudo-element selectors in a browser



Accessibility guideline

- Apply the same formatting to the :hover and :focus pseudo-classes for an element. That way, those who can't use the mouse will have the same experience as those who can.

Description

- *Pseudo-classes* are predefined classes that apply to specific conditions. In contrast, *pseudo-elements* let you select a portion of text.

Figure 4-9 How to code pseudo-class and pseudo-element selectors

How to work with Cascading Style Sheets

The term *Cascading Style Sheets* refers to the fact that more than one style sheet can be applied to a single web page. Then, if two or more declarations for the same property are applied to the same element, the cascade order and other rules determine which declaration takes precedence.

How the cascade rules work

Before you can understand the cascade rules, you need to know that users can create user style sheets for their browsers that provide default style rules for their web pages. Because most users don't create user style sheets, this usually isn't an issue. But some users do. For instance, users with poor vision often create user style sheets that provide for large font sizes. In that case, you need to consider how the user style sheets could affect your web pages.

You should also know how to identify one of your declarations as important so it has precedence over other declarations. To do that, you code "!important" as part of the declaration. This is shown by the example in figure 4-10.

With that as background, this figure lists the five levels of the cascade order from highest to lowest. As you can see, the important declarations in a user style sheet override the important declarations in a web page, but the normal declarations in a web page override the normal declarations in a user style sheet. Below these declarations are the default declarations in the web browser.

What happens if an element has more than one declaration applied to it at the same level? To start, the declaration with the highest specificity takes precedence, and this figure shows you which parts of a selector are more specific. For instance, the .speakers selector is more specific than the li selector because a class is more specific than an element.

If a selector contains multiple parts, the additional parts add to that selector's specificity. This means that a selector with an element and a class is more specific than a selector with just a class. For instance, the p.highlight selector is more specific than the .highlight selector. As a result, p.highlight takes precedence.

If that doesn't settle the conflict, the declaration that's specified last is applied. Earlier in this chapter, for example, you learned that if you provide two or more external style sheets for a document, the style rules in each style sheet will override the style rules in the preceding style sheets. This notion also applies if you accidentally code two style rules for the same element in a single style sheet. Then, the one that is last takes precedence. In addition, inline styles take precedence over the styles in an embedded style sheet, and embedded styles take precedence over the styles in an external style sheet.

How to identify a declaration as important

```
.highlight {
    font-weight: bold !important;
}
```

The cascade order for applying CSS style rules

Search for the style rules that apply to an element in the sequence that follows and apply the style rule from the first group in which it's found:

- !important declarations in a user style sheet
- !important declarations in a web page
- Normal declarations in a web page
- Normal declarations in a user style sheet
- Default declarations in the web browser

If more than one style rule at a cascade level is applied to an element...

- Use the style rule with the highest specificity. For example, the p.highlight selector is more specific than the .highlight selector.
- If the specificity is the same for two or more style rules in a group, use the style rule that's specified last.

How to determine the specificity of a selector

- An id is the most specific.
- A class, attribute selector, or pseudo-class selector is less specific.
- An element or pseudo-element selector is least specific.

Description

- When two or more style rules are applied to an HTML element, CSS uses the *cascade order* and rules shown above to determine which style rule to apply.
- A user can create a *user style sheet* that provides a default set of style rules for web pages. Users with poor vision often do this so the type for a page is displayed in a large font.
- Since most users don't create user style sheets, you usually can control the way the style rules are applied for your websites. But you should keep in mind how your web pages could be affected by user style sheets.
- If you want to create or remove a user style sheet for a browser, you can search the Internet for the procedures that your browser requires.

Figure 4-10 How the cascade rules work

How to use the developer tools to inspect the styles that have been applied

If you have problems with cascading styles when you're testing a web page, you can use the developer tools for your browser to find out exactly what's happening, as shown in figure 4-11. If you're using Chrome, Internet Explorer, Edge, or Firefox, you can access these tools by pressing the F12 key. You can also access the developer tools from these browsers as well as from Opera and Safari by right-clicking on a page and selecting Inspect or Inspect Element.

In this figure, the developer tools are displayed in a panel below the web page in a Chrome browser. This is the same page that's shown in figure 4-17, which you'll study at the end of this chapter.

In the left pane of the developer tools panel, you can see all of the HTML elements of the page. To expand or collapse a group of elements, you click on the symbol before the element. In this example, you can see that the html, body, and main elements have been expanded.

To inspect the styles for an element, you click on the element in the Elements pane. Or, you can click on the inspect icon at the left of the toolbar for the developer tools (the one with an arrow pointing to a square on it), and then click on the element in the web page. In this case, the user has clicked on the h1 element in the Elements pane.

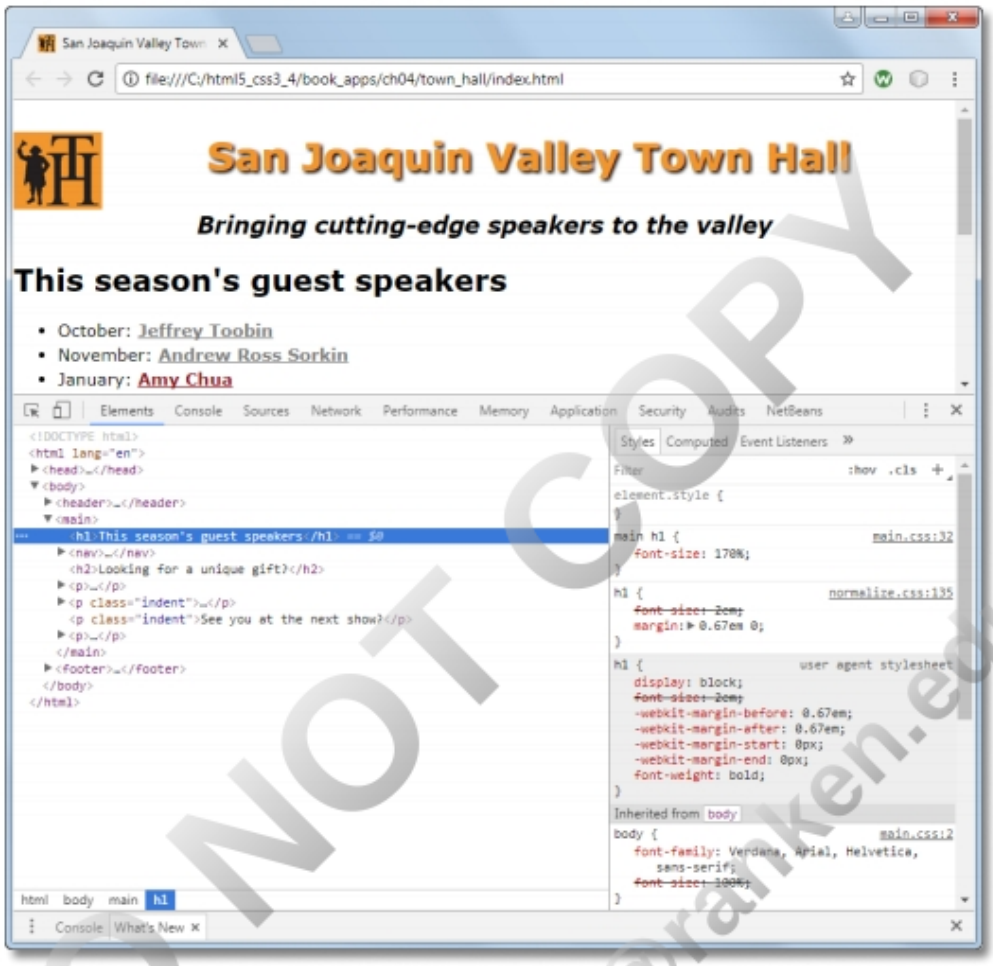
When an element is selected, its styles are shown in the Styles pane at the right side of the developer tools panel. This pane shows all of the styles that have been applied to the element, from the first styles at the bottom of the pane to the last styles at the top. In this example, the bottom of the pane shows the styles that are inherited from the body element. Here, the font-size style has been crossed out because it has been overridden. Next, the shaded group shows the styles that are applied by the user agent style sheet, which is the one provided by the browser. Here again, the font-size style has been overridden.

The last two groups are for the normalize style sheet and the main style sheet. This shows that the main style sheet sets the final value of the font-size property to 170%. This overrides the value for this property set by the preceding groups. However, the margin property for this element in the normalize style sheet hasn't been overridden, so a top and bottom margin of .67em is retained.

If you scroll to the bottom of the Styles pane, you can see a graphic representation of this element that shows the exact size of these margins. This graphic also shows the width and height of the element, as well as any padding and border that's been applied to it. You'll learn about these properties in the next chapter. For now, just realize that the developer tools can be a valuable resource for identifying problems with the styles for these properties as well as any other properties in the style sheets for a page.

You should also realize that you can do more than just inspect styles using the developer tools. For instance, you can also change the generated HTML to see how that affects the page. And you can edit the styles that are applied to a page. For more information on these and other features of the developer tools, please see the help information for your browser.

How Chrome's developer tools show the effects of cascading styles



How to use Chrome's developer tools

- To display the panel for the tools, press the F12 key.
- To inspect the styles that have been applied to an element, click on the element in the Elements pane at the left side of the developer tools panel. Or, click on the inspect icon at the left of the toolbar for this panel, and then click on an element in the web page.
- The styles that have been applied to the selected element are displayed in the Styles pane at the right side of the developer tools panel.

Description

- Most modern browsers provide developer tools that can be accessed by pressing the F12 key or by right-clicking on the page and selecting Inspect or Inspect Element.
- One of the primary uses of the developer tools is to inspect the styles that have been applied to an element, including how the styles in one style sheet have overridden inherited styles or the styles in another style sheet.

Figure 4-11 How to use the developer tools to inspect styles

How to work with text

Now that you know how to select the elements that you want to format, you're ready to learn how to use CSS to apply that formatting. To start, you'll learn how to style fonts and format text.

How to set the font family and font size

Figure 4-12 shows how to set the *font family* and font size for text elements. The table at the top of this figure lists the five generic font families, and the examples below that table show what typical fonts in these families look like.

When you develop a website, your primary font family should be a sans-serif font family. That's because sans-serif fonts are easier to read in a browser than the other types of fonts, including serif fonts, even though serif fonts have long been considered the best for printed text. You can also use serif and monospace fonts for special purposes, but you should avoid the use of cursive and fantasy fonts.

When you code the values for the font-family property, you code a list of the fonts that you want to use. For instance, the first example in this figure lists Arial, Helvetica, and sans-serif as the fonts, and sans-serif is a generic font name. Then, the browser will use the first font in the list that is available to it. But if none of the fonts are available, the browser will substitute its default font for the generic font that's coded last in the list.

If the name of a font family contains spaces, like "Times New Roman", you need to enclose the name in quotation marks when you code the list. This is illustrated by the second and third examples in the first group.

To set the font size for a font, you use the font-size property as illustrated by the second group of examples. For this property, we recommend relative measurements so the users will be able to change the font sizes in their browsers.

When you use a relative measurement, it's relative to the parent element. For example, the second rule in the second set of examples will cause the font to be 150% larger than its parent element. So if the parent element is 16 points, this element will be 24 points. Similarly, the third rule specifies 1.5 ems so it will also be 150% of the parent font.

The next example shows how the font family and font size can be set in the body element. Here, the default font family for the browser is changed to a sans-serif font. This overrides the Times New Roman font that's used by most browsers. In addition, the default font size is set to 100% of the browser's default size. Although this doesn't change the font size, it does make the size relative to the browser's default size, which is usually 16 pixels. Then, if the user changes the browser's font size, that change is reflected in the web page.

Like colors, the font properties that you set in an element are inherited by all of its descendents. Note, however, that it's not the relative value that's inherited when you use a relative measurement for the font size. Instead, it's the actual size that's calculated for the font. If, for example, the default font size is 16 pixels and you set the font-size property to 125%, the size of the element and any descendents will be 20 pixels.

The five generic font families

Name	Description
`serif`	Fonts with tapered, flared, or slab stroke ends.
`sans-serif`	Fonts with plain stroke ends.
`monospace`	Fonts that use the same width for each character.
`cursive`	Fonts with connected, flowing letters that look like handwriting.
`fantasy`	Fonts with decorative styling.

Examples of the five generic font families

Times New Roman is a serif font. It is the default for most web browsers.

Arial is a sans-serif font that is widely used, and sans-serif fonts are best for web pages.

Courier New is a monospace font that is used for code examples.

Lucida Handwriting is a cursive font that is not frequently used.

Impact is a fantasy font that is rarely used.

How to specify a font family

```
font-family: Arial, Helvetica, sans-serif;
font-family: "Times New Roman", Times, serif;
font-family: "Courier New", Courier, monospace;
```

How to specify the font size

```
font-size: 12pt;          /* in points */
font-size: 150%;          /* as a percent of the parent element */
font-size: 1.5em;         /* same as 150% */
```

A font-family rule in the body element that is inherited by all descendants

```
body {
    font-family: Arial, Helvetica, sans-serif;
    font-size: 100%; }
```

A font-family rule in a descendent that overrides the inherited font family

```
p { font-family: "Times New Roman", Times, serif; }
```

Description

- The fonts specified for the font-family property are searched in the order listed. If you include a font name that contains spaces, the name must be enclosed in quotes.
- If you specify a generic font last and the web browser can't find any of the other fonts in the list, it will use its default font for the generic font that you specified.
- The font properties that you set for an element are inherited by all of its descendants.
- If you use relative font sizes, the users will be able to vary the sizes by using their browsers. If you use pixels, the font size will vary based on the screen resolution.

Figure 4-12 How to set the font family and font size

How to set the other properties for styling fonts

The table in figure 4-13 summarizes the other properties that you can use for styling a font. Like the font-family and font-size properties, these properties are also inherited by their descendents.

In this figure, the first group of examples shows how italics and small caps can be applied to a font by using the font-style and font-variant properties. Then, the second group of examples shows how font weights can be applied to a font. In most cases, you'll use the bold keyword to boldface a font. But if you want to use varying degrees of boldness, you can use multiples of 100. You can also specify the boldness of a font relative to the boldness of the parent font by using the lighter or bolder keyword as shown in the last example.

The third group of examples shows how to set the line height for a font. This property lets you increase or decrease the amount of vertical space that's used for a font. If, for example, you set the line height to 14 points for a font that is set to 12 points, there will be two extra points of space for the font. This space will be divided equally above and below the font. This provides extra spacing for block elements that are displayed on more than one line.

Like the font-size property, it's usually better to set the line-height property with a relative measurement like a percent or ems. That way, all modern browsers will be able to adjust the line height relative to the font size. The first three examples illustrate that. The fourth example is similar, but it specifies a number that is used to calculate the line height.

Note that when you use a percent or ems as in the second and third examples, the actual line height will be inherited by descendent elements. When you use a number, however, the number itself will be inherited. If a descendent element has a smaller font size, then, the line height will also be smaller, which is usually what you want.

Often, you code each of the font properties that you've just learned about in a separate rule. However, you can also use the *shorthand property* for fonts that's shown in this figure. If you look at the syntax for this property, you can see that it provides for all six font properties. In this syntax summary, the properties in brackets [ ] are optional, which means that only the font-size and font-family are required.

When you use this property, you code the font properties separated by spaces without coding the property names. You combine the font-size and line-height properties separated by a slash. And you separate the list of fonts for the font-family property with commas. This is illustrated by the three examples after the syntax summary.

Here, the first declaration includes all the font properties except for font-variant. The second declaration includes the font-variant, font-size, and font-family properties. And the third declaration includes the font-size, line-height, and font-family properties.

Other properties for styling fonts

Property	Description
`font-style`	A keyword that determines how the font is slanted: normal, italic, and oblique.
`font-weight`	A keyword or number that determines the boldness of the font: normal, bold, bolder, lighter, or multiples of 100 from 100 through 900, with 400 equivalent to normal. Bolder and lighter are relative to the parent element.
`font-variant`	A keyword that specifies whether small caps will be used: normal and small-caps.
`line-height`	A relative or absolute value or a number that specifies the amount of vertical space for each line. The excess space is divided equally above and below the font.

How to specify font styles and variants

```
font-style: italic;
font-style: normal;          /* remove style */
font-variant: small-caps;
```

How to specify font weights;

```
font-weight: 700;
font-weight: bold;           /* same as 700 */
font-weight: normal;         /* same as 400 */
font-weight: lighter;        /* relative to the parent element */
```

How to specify line height

```
line-height: 14pt;
line-height: 140%;
line-height: 1.4em;          /* same as 140% */
line-height: 1.4;            /* same as 140% and 1.4em */
```

The syntax for the shorthand font property

```
font: [style] [weight] [variant] size[/line-height] family;
```

How to use the shorthand font property

```
font: italic bold 14px/19px Arial, sans-serif;
font: small-caps 150% "Times New Roman", Times, serif;
font: 90%/120% "Comic Sans MS", Impact, sans-serif;
```

Description

- You can set the font-style, font-weight, and font-variant properties to a value of "normal" to remove any formatting that has been applied to these properties.
- The line-height property determines the spacing between lines within a block element.
- If you specify just a number for the line-height property, the font size is multiplied by that value to determine the line height, and the multiplier is inherited by child elements. If you specify an absolute or relative size for the line-height property, the actual line height is inherited by child elements.
- You can use the shorthand property for a font to set all six font properties with a single rule. When you use this property, the font-size and font-family properties are required.

Figure 4-13 How to set the other properties for styling fonts

How to set properties for formatting text

Figure 4-14 presents four of the properties for formatting text. You can use the text-indent property to indent the first line of text in a paragraph. When you set this property, it usually makes sense to use a relative unit of measure such as ems. That way, if the size of the current font changes, the indentation will also change.

To align text horizontally, you can use the text-align property. By default, most elements are left-aligned, but you can use the "center", "right", or "justify" values to change that. Note, however, that when you justify text, the spacing between words is adjusted so the text is aligned on both the left and right sides. Since this makes the text more difficult to read, justified text should be avoided.

You can also align inline elements vertically. To illustrate, suppose that you code a span element within a paragraph, and you code a style rule that sets the font size for the text in the span element so it's smaller than the text in the paragraph. Then, the vertical-align property determines how the span element is aligned relative to its parent element. If, for example, you set the vertical-align property to "text-bottom", the bottom of the text in the span element will be aligned with the bottom of the text in the paragraph. Another alternative is to specify a relative or absolute value for this property to determine how far above or below its normal position the element should be displayed.

The example in this figure shows how to use the text-indent and text-align properties. Here, the paragraph below the heading is indented by 2 ems, and the paragraph that contains the copyright information is right-aligned.

You can use the text-decoration property to display a line under, over, or through text. However, this property has limited value for two reasons. First, you usually shouldn't underline words that aren't links. Second, you can use borders as shown in the next chapter to put lines over and under a block element, and that gives you more control over the lines.

If you want to remove any text decoration that has been applied to an element, you can specify a value of "none" for this property. For example, the text-decoration property of an <a> element is set to "underline" by default. If that's not what you want, you can set this property to "none". You'll see how this works in chapter 7.

Properties for indenting, aligning, and decorating text

Property	Description
`text-indent`	A relative or absolute value that determines the indentation for the first line of text. This property is inherited.
`text-align`	A keyword that determines the horizontal alignment of text. Possible values are left, center, right, and justify. This property is inherited.
`vertical-align`	A relative or absolute value or a keyword that determines the vertical alignment of text. Possible keywords are baseline, bottom, middle, top, text-bottom, text-top, sub, and super.
`text-decoration`	A keyword that determines special decorations that are applied to text. Possible values are underline, overline, line-through, and none.

The HTML for a web page

```
<header>
    <h1>San Joaquin Valley Town Hall</h1>
</header>
<main>
    <p>Welcome to San Joaquin Valley Town Hall. We have some
       fascinating speakers for you this season!</p>
</main>
<footer>
    <p>&copy; Copyright 2018 San Joaquin Valley Town Hall.</p>
</footer>
```

CSS that specifies a text indent and horizontal alignment

```
body {
    font-size: 100%;
    margin: 2em; }
h1 { font-size: 180%; }
main p { text-indent: 2em; }
footer p {
    font-size: 80%;
    text-align: right; }
```

The HTML in a web browser

Description

- The text-indent and text-align properties are often used with text, and the vertical-align property is often used with tables.
- The text-decoration property is often set to "none" to remove the underlines from links.

Figure 4-14 How to set properties for formatting text

How to use CSS3 to add shadows to text

Before CSS3, you had to use an image to display text with shadows. But now, CSS3 offers the text-shadow property for this purpose. That makes it much easier to provide shadows.

Figure 4-15 shows how to use this property. As the syntax shows, you can set four parameters for it. The first one specifies how much the shadow should be offset to the right (a positive value) or left (a negative value). The second one specifies how much the shadow should be offset down (a positive value) or up (a negative value). The third one specifies how big the blur radius for the shadow should be. And the fourth one specifies the color for the shadow.

The first example shows text with a shadow that is 4 pixels to the right and down, with no blur and with the shadow the same color as the text. The result is shown in the browser.

In contrast, the shadow for the heading in the second example is offset to the left and up by 2 pixels, with a blur radius of 4 pixels and with the shadow in red. Since the heading is in blue, this provides an interesting effect.

As this figure shows, this property is supported by all modern browsers. But if a browser doesn't support this property, it is simply ignored so no harm is done: the heading is just displayed without the shadow. That's why you can start using this property right away.

When you use this property, though, remember the visually-impaired. If the offsets or blur are too large, the shadow can make the text more difficult to read. On that basis, both examples in this figure should probably be toned down so they're easier to read.

The syntax of the text-shadow property

```
text-shadow: horizontalOffset verticalOffset blurRadius shadowColor;
```

Two examples

The h1 element

```
<h1>San Joaquin Valley Town Hall</h1>
```

The CSS

```
h1 {
    color: #ef9c00;
    text-shadow: 4px 4px; }
```

The heading in a browser

Different CSS for the same h1 element

```
h1 {
    color: blue;
    text-shadow: -2px -2px 4px red; }
```

The heading in a browser

Accessibility guideline

- Remember the visually-impaired. Too much shadow or blur makes text harder to read.

Description

- Positive values offset the shadow to the right or down. Negative values offset the shadow to the left or up.
- The blur radius determines how much the shadow is blurred.
- The text-shadow property is supported by all modern browsers.
- If this property isn't supported by a browser, it is ignored so there's no shadow, which is usually okay.

Figure 4-15 How to use CSS3 to add shadows to text

How to float an image so text flows around it

In chapters 5 and 6, you'll learn everything you need to know about setting margins and floating an image so the text flows around it. But just to get you started with this, figure 4-16 shows how you can float the logo for a header to the left so the headings flow to its right.

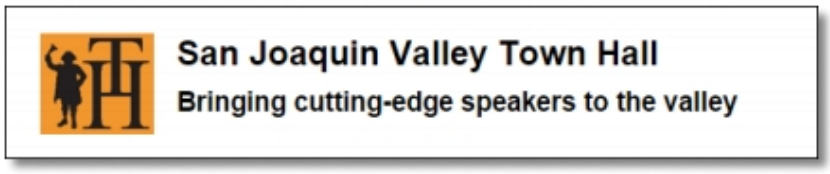
In the HTML for the example, you can see an img element, an h1 element, and an h2 element. Then, in the CSS for the img element, the float property is set to left and the right-margin property is set to 1em. The result is that the two headings flow to the right of the image as shown in the first browser example.

You should know, however, that this depends on the size of the image. Because the width of the image is set to 80 pixels for the first example, both headings flow to its right. But if the width is reduced to 40 pixels as in the second example, the height is also reduced. Then, the second heading is under the image because the image is too short for the heading to flow to its right.

This shows that you're going to have to fiddle with the image size to get this to work right for the time being. But in the next two chapters, you'll learn the right ways to get this result.

If you want to stop elements from flowing to the right of a floated element, you can use the clear property that's shown in this figure. Here, the clear property is used for the main element, and it stops the flow around an element that has been floated to its left. If the main element follows a header, the flow of the text will stop before any of the elements within the main element are displayed.

An image that has been floated to the left of the headings that follow



The HTML

```
<img src="images/logo.gif" alt="Town Hall Logo" width="80">
<h1>San Joaquin Valley Town Hall</h1>
<h2>Bringing cutting-edge speakers to the valley</h2>
```

The CSS

```
img {
    float: left;
    margin-right: 1em;
}
```

The page if the width of the image is reduced to 40

The property that will stop the floating before a subsequent element

```
main { clear: left; }
```

Description

- To float an image, you use the float property, and to set the margins around it, you use the margin property.
- In chapters 5 and 6, you'll learn how to set margins and float elements, but this will give you an idea of how you can use an image in a header.
- When you float an image to the left, the block elements that follow it fill the space to the right of it. When the elements that follow get past the height of the image and its top and bottom margins, they flow into the space below the element.
- You can use the clear property to stop an element from flowing into the space alongside a floated element.
- For now, you can experiment with the size of the image to get the effect that you want, but in the next two chapters you'll learn the right ways to get the same results.

Figure 4-16 How to float an image so text flows around it

A web page that uses external style sheets

Now that you've learned how to code selectors and how to format text, you're ready to see a web page that uses these skills.

The page layout

Figure 4-17 presents a web page that uses two external style sheets. First, it uses the normalize.css style sheet that you learned about earlier in this chapter. Second, it uses a style sheet with styles that are specific to this page.

If you study this web page, you can see that the CSS in the style sheets has been used to change the default font to a sans-serif font, to apply colors to some of the headings and text, to center the headings, to apply shadows to the text in the first heading, to add line height to the items in the unordered list, to apply boldfacing to portions of text, and to right-align the footer. Overall, this formatting makes the page look pretty good.

In terms of typography, though, this web page needs to be improved. For instance, there should be less space after "San Joaquin Valley Town Hall", less space after "This season's guest speakers", less space after "Looking for a unique gift?", and less space between the first three paragraphs. There should also be space at the left and right sides of the page. To make these adjustments, though, you need to know how to use the margin and padding properties that are part of the CSS box model. So that's what you'll learn first in the next chapter. Once you learn how to use those properties, you'll be able to get the typography just the way you want it.

A web page that uses some of the styles presented in this chapter

Description

- This web page uses an external style sheet to apply the styles that are illustrated.
- A sans-serif font has been applied to all of the text in this document because that's the most readable type of font for web pages. Also, relative font sizes have been applied to the elements on the page.
- Italics and boldfacing have been applied to portions of the text, the copyright information has been right-aligned, and colors have been applied to two of the headings.
- Colors have also been applied to the <a> tags in the unordered list. Because the first two lectures have passed, the first two links in the list have the color gray applied to them.

What's wrong with the typography in this web page

- The spacing above and below the block elements and around the body of the page should be improved. In the next chapter, you'll learn how to do that by using the margin and padding properties.

Figure 4-17 The page layout for a web page that uses an external style sheet

The HTML file

Figure 4-18 presents the HTML for the web page in figure 4-17. In the head section, you can see the two link elements that refer to the external style sheets that are used for formatting this web page. They are both in the styles folder.

In the body section, the header and the first part of the main element are the same as they were in the example at the end of the last chapter with one exception. That is, the <a> elements in the first and second <li> elements have class attributes that assign them to the "date_passed" class. This class name will be used to apply the color gray to those items because this example assumes that the dates for those events have already passed, and the gray color is intended to indicate that to the user.

After the nav element, this HTML includes another h2 element and three new <p> elements. The second and third <p> elements have their class attributes set to "indent". Then, this class will be used as the CSS selector for indenting these paragraphs.

The HTML file for the web page

```
<!DOCTYPE HTML>

<html lang="en">
<head>
    <title>San Joaquin Valley Town Hall</title>
    <meta charset="utf-8">
    <link rel="shortcut icon" href="images/favicon.ico">
    <link rel="stylesheet" href="styles/normalize.css">
    <link rel="stylesheet" href="styles/main.css">
</head>

<body>
    <header>
        <img src="images/logo.gif" alt="Town Hall Logo" width="80">
        <h2>San Joaquin Valley Town Hall</h2>
        <h3>Bringing cutting-edge speakers to the valley</h3>
    </header>
    <main>
        <h1>This season's guest speakers</h1>
        <nav>
            <ul>
              <li>October: <a class="date_passed"
                  href="speakers/toobin.html">Jeffrey Toobin</a></li>
              <li>November: <a class="date_passed"
                  href="speakers/sorkin.html">Andrew Ross Sorkin</a></li>
              <li>January: <a href="speakers/chua.html">
                  Amy Chua</a></li>
              <li>February: <a href="speakers/sampson.html">
                  Scott Sampson</a></li>
              <li>March: <a href="speakers/eire.html">
                  Carlos Eire</a></li>
              <li>April: <a href="speakers/tynan.html">
                  Ronan Tynan</a></li>
            </ul>
        </nav>

        <h2>Looking for a unique gift?</h2>
        <p>Town Hall has the answer. For only $100, you can get a book of
           tickets for all of the remaining speakers. And the bargain
           includes a second book of tickets for a companion.</p>
        <p class="indent">Or, for $50, you can give yourself the gift of
           our speakers, and still get an extra ticket for a companion, but
           for just one of the events.</p>
        <p class="indent">See you at the next show?</p>

        <p><em>Contact us by phone</em> at (559) 555-1212 for ticket
           information.</p>
    </main>
    <footer>
        <p>&copy; Copyright 2018 San Joaquin Valley Town Hall.</p>
    </footer>
</body>
</html>
```

Figure 4-18 The HTML file for the web page

The CSS file

Figure 4-19 shows the CSS file for the web page. To start, notice the way that the code in this file is structured. First, the style rules for specific elements are presented. In effect, this sets the default formatting for these elements. Then, these style rules are followed by style rules that are specific to the header, main, and footer elements. The declarations in these style rules override the ones for the elements because they're more specific. For instance, the font size for the main h1 selector overrides the font size for the body selector.

You might also note that many of the style rules that contain a single declaration are coded on a single line. This of course is a valid way to code these style rules because white space is ignored. For style rules that require more than one declaration, though, each declaration is coded on a single line to make the declarations easier to read.

In the style rule for the body element, the two properties specify the font-family and font-size. Because these properties are inherited, they become the defaults for the document. Notice too that the font size is specified as 100 percent. That means the actual font size will be determined by the default font size for the browser. The font sizes for the headings are also specified as percents, so they will be based on the size that's calculated for the body element.

In the style rules for the <a> element, the hover and focus pseudo-classes are set to the same color. Then, in the style rule for the unordered list, the line-height property is specified to increase the spacing between the items. Similarly, the font weight is specified for the em element so it is boldfaced. However, this element will also be italicized because that's its normal behavior. To remove the italics, you would have to code a font-style property with normal as its value.

In the style rules for the header, you can see that the image is floated to the left so the elements that follow will appear to the right. If you look back at the web page in figure 4-17, though, you'll see that only the h2 heading flows to the right of the image. That's because the image is too short for the h3 element to flow to its right. Then, although both the h2 and h3 elements are centered, the h2 element appears centered in the header but the h3 element appears centered on the page. You can also see that the h2 heading has a 2 pixel, black shadow to its right and down with 2 pixels of blur.

In the style rules for the main element, you can see that the clear property is used to stop the flow of the text around the image that has been floated left. You can also see that the elements in the "indent" class will be indented 2 ems, and the elements in the "date_passed" class will be gray.

Of course, you can code the CSS in other ways and get the same results. For instance, you could code this style rule for the h2 and h3 elements in the header:

```
header h2, header h3 { text-align: center; }
```

Then, you could drop the text-align properties from the header h2 and header h3 style rules that follow. You could also code the selectors for the date_passed and indent classes so they're less specific since they aren't used outside the main element. If you code the selectors so they're more specific, though, they're less likely to cause problems if you add style rules later on.

The CSS file for the web page

```
/* the styles for the elements */
body {
    font-family: Verdana, Arial, Helvetica, sans-serif;
    font-size: 100%;
}

a { font-weight: bold; }
a:link { color: #931420; }
a:visited { color: #f2972e;}
a:hover, a:focus { color: blue; }

ul { line-height: 1.5; }
li, p { font-size: 95%; }
em { font-weight: bold; }

/* the styles for the header */
header img { float: left; }
header h2 {
    font-size: 220%;
    color: #f2972e;
    text-align: center;
    text-shadow: 2px 2px 2px black;
}
header h3 {
    font-size: 130%;
    font-style: italic;
    text-align: center;
}

/* the styles for the main content */
main { clear: left; }
main h1 { font-size: 170%; }
main h2 { font-size: 130%; }

main p.indent { text-indent: 2em; }
main a.date_passed { color: gray; }

/* the styles for the footer */
footer p {
    font-size: 80%;
    text-align: right;
}
```


Figure 4-19 The CSS file for the web page

Perspective

At this point, you should know how to code all types of selectors, and you should know how to code the CSS properties for formatting text. That gets you off to a good start with CSS, but there's still a lot to learn.

So, in the next chapter, you'll learn how to use the CSS box model. You use that model to set the margins, padding, and borders for the elements in your pages so your typography looks the way you want it to. Then, in chapter 6, you'll learn how to use CSS for page layout.

Terms

external style sheet
embedded style sheet
inline style
shiv
shim
absolute unit of measure
relative unit of measure
absolute measurement
relative measurement
RGB value
hexadecimal (hex) value
inherited property
RGBA value
HSL value
HSLA value
universal selector
type selector
id selector
class selector
relational selector
descendant selector
adjacent sibling selector
child selector
general sibling selector
attribute selector
pseudo-class selector
pseudo-element selector
cascade order
user style sheet
font family
shorthand property

Summary

- If you're going to use a style sheet for more than one HTML document, it's usually best to use an *external style sheet*. However, you can also apply CSS by embedding a style sheet in the HTML or by using the style attribute of an element.
- To provide cross-browser compatibility with older versions of Internet Explorer when you use the HTML5 semantic elements, you can add a script element that accesses a JavaScript *shiv* to the head element of each HTML document.
- To make sure that all HTML elements are rendered the same in all modern browsers, you can include the normalize.css style sheet in your web pages.
- You can use *absolute measurements* like pixels or *relative measurements* like ems or percents to specify the CSS properties for sizes. For fonts, it's better to use relative measurements so the user can change the font sizes by changing the browser's default font size.

- Most graphic designers use hex for the RGB values that represent the colors that they want because that gives them the most control. However, most browsers also support 16 standard color names like red and blue.
- CSS3 lets you use *RGBA*, *HSL*, and *HLSA* values to specify colors. This gives the web designer more control over colors and transparency. CSS3 also provides 147 more keywords for colors.
- You can code CSS selectors for element types, ids, classes, relations, and attributes. You can also code selectors for combinations of these items.
- A *pseudo-class selector* can be used to apply CSS formatting when certain conditions occur or have occurred, like when the mouse hovers over an element or the focus is on an element. A *pseudo-element selector* lets you select a portion of text.
- If more than one style sheet is applied to an HTML document and two or more style rules apply to the same element, the *cascade order* determines which style rule takes precedence. The first four levels of this order are: the important declarations in a *user style sheet* for the browser; the important declarations in a web page; the normal declarations in a web page; and the normal declarations in a user style sheet.
- If more than one style rule in a cascade level is applied to an element, the style rule with the highest specificity is used. But if the specificity is the same for two or more style rules in a cascade level, the style rule that's specified last is used.
- The developer tools for a modern browser let you inspect all of the styles that have been applied to an element, including the styles that have been overridden.
- The default font for most browsers is a 16-pixel, serif font. However, because sans-serif fonts are easier to read in a browser, you normally change the font family. It's also good to change the font size to a relative measurement like a percent of the default font size.
- The colors and font properties of a parent element are *inherited* by the child elements. But those properties can be overridden by the style rules for the children.
- The *shorthand property* for fonts can be used to apply all six of the font properties to an element: font-family, font-size, font-style, font-weight, font-variant, and line-height.
- Text properties can be used to indent, align, and decorate the text in a block element like a heading or paragraph. CSS3 also provides for adding shadows to text.
- You can use the float property of an image to float the image to the left. Then, the block elements that follow in the HTML flow to its right. To stop the flow, you can code the clear property for an element.

Exercise 4-1 Format the Town Hall home page

In this exercise, you'll format the home page that you built in exercise 3-1 by using the skills that you've learned in this chapter. When you're through, the page should look like this.

Open the index.html page and update the head section

1. Use your text editor to open the HTML file that you created for exercise 3-1:
   `c:\html5_css3_4\exercises\town_hall_1\index.html`
2. Use your text editor to open this HTML template file:
   `c:\html5_css3_4\exercises\town_hall_1\templates\basic.html`
   Then, copy the second and third link elements from the head section to the clipboard, switch to the index.html file, and paste these elements at the end of the head section.
3. Note that the first link element you just copied into the head is for the normalize.css style sheet in the styles subfolder. Next, complete the href attribute in the second link element so it refers to the main.css file in the styles subfolder. Then, close the template.

Open the main.css file and format the header

4. Use your text editor to open this CSS file:
   `c:\html5_css3_4\exercises\town_hall_1\styles\main.css`
   Note that this file contains some of the CSS code that you'll need, including the style rule that specifies the font family and font size for the body, the style rule that floats the image in the header, and the style rule that clears the floating in the main element.
5. Add two style rules for the header to the style sheet. The first one should be for the h2 element, and it should set the font size to 170%, set the color to #800000, and indent the heading 30 pixels. The second one should be for the h3 element, and it should set the font size to 130%, set the font style to italic, and indent the heading 30 pixels.
6. Test the HTML page in Chrome to make sure that the style sheets have been linked properly, the image has been floated, and the headings have been formatted correctly. If necessary, make corrections and test again.

Format the main element and the footer

From now on, test each change right after you make it.

7. Add a style rule for the h1 elements within the main element that sets the font size to 150%.
8. Add a style rule for the h2 elements within the main element that sets the font size to 130% and the color to #800000.
9. Add a style rule for the h3 elements within the main element that sets the font size to 105%.
10. Add a style rule that italicizes any link that has the focus or has the mouse hovering over it.
11. Add a style rule that centers the <p> tag in the footer.

Use the developer tools to review the styles for the page

12. Display the page in Chrome, and then press the F12 key to display the developer tools. Next, expand the main element in the Elements pane and click on one of the h2 elements.
13. Review the styles for the h2 element in the Styles pane, and notice how the font-family style for the html element in the normalize style sheet is overridden by the font-family style for the body element in the main style sheet. Also notice how the font-size style for the body element in the main style sheet and the h2 element in the user agent style sheet are overridden by the font-size style for the main h2 element in the main style sheet.
14. Click the icon in the developer tools toolbar that has a square with an arrow pointing to it on it, and then click on the h2 element in the header to see that it's now selected in the Elements pane.
15. Review the styles for this h2 element to see that they're similar to the styles for the main h2 element. However, the font size for this element is larger and it has a text indent.

16. When you're done with the developer tools, close the panel by clicking the icon with an "X" on it in the upper right corner.

Add the finishing touches and test in IE

17. Add a text shadow to the 75th in the second heading in the header. To do that, enclose the *75th* in the HTML in an em or span element and give that element a class attribute with a value of "shadow". Then, create a style rule that uses a class selector (.shadow) for that class, and code a declaration that adds a shadow to the text with #800000 as the color of the shadow.
18. Test the page in IE. There the text shadow won't work if you're using a version before version 10. But if you see any other formatting problems, make the corrections and test again in both Chrome and IE.