

## 3.11 External CSS Files

### Overview

In general, splitting up a big thing into smaller parts makes the thing easier to understand. To improve understandability, you should consider moving your CSS rules to an external file. There are two steps necessary to tie a web page to an external file that contains CSS rules. First, for the external file to be recognized as a CSS file, the external file must be named with a `.css` extension. For example, in an upcoming web page, we'll use the filename `pumpkinPatch.css`. Second, for the web page to access a CSS file's CSS rules, the web page must use a `link` element in the web page's head container. The `link` element is a void element, so it's comprised of just one tag, with no end tag. Here's the syntax:

```
<link rel="stylesheet" href="name-of-external-file" >
```

In this code fragment, note the `href="name-of-external-file"` attribute-value pair. The W3C does not specify what `href` stands for, but the most common belief is that it stands for “hypertext reference.” The `href` attribute's value specifies the name of the file that holds the CSS rules (e.g., `pumpkinPatch.css`).

Note the `rel="stylesheet"` attribute-value pair. `rel` stands for “relationship,” and its value tells the browser engine what to do with the `href` file. Having a `rel` value of `stylesheet` tells the browser engine to look for CSS rules in the `href` file and apply them to the current web page.

To justify the extra work of adding a `link` element to handle an external CSS file, typically an external CSS file will be nontrivial. That means the file will contain at least five CSS rules (usually a lot more), or it will be shared by more than one web page.

Why is sharing an external CSS file helpful? With a shared external CSS file, it's easy to ensure that all the web pages on your site follow the same common CSS rules. And if you want to change those rules, you change them in one place, in the external file, and the change affects all the web pages that share the external file.

External CSS files used to be referred to as “external style sheets,” but the W3C no longer uses that term, and we won’t use it either. However, you should recognize the term “external style sheets” because you’ll hear it being used every now and then.

## Example

Let’s put what you’ve learned into practice by examining the source code for a modified version of the Pumpkin Patch web page—a version that uses an external CSS file instead of a `style` container. See **FIGURE 3.9**, which shows the `head` container for the modified Pumpkin Patch web page. Note that there’s no `style` container; instead there’s a `link` element that connects to an external CSS file.

See **FIGURE 3.10**. It shows the source code for the `pumpkinPatch.css` external CSS file, which gets loaded into the new Pumpkin Patch web page. The external CSS file’s CSS rules are identical to those found in the original Pumpkin Patch web page, so, as you’d expect, there is no difference in how the two web pages render.

Normally, external CSS files are rather sparse. They include CSS rules, with blank lines to separate logical groups of rules, and that’s pretty much it. Optionally, you can include comments to explain nonintuitive characteristics of the CSS file. In the previous chapters, you’ve learned to include a `meta author` element at the top of your HTML files, so other people in your company know whom to go to when questions arise. Likewise, for each external CSS file, you should include a comment at the top that shows the author’s name. Here’s an example author’s-name comment, copied from the top of the `pumpkinPatch.css` source code:

```
/* John Dean */
```

```
<head>
<meta charset="utf-8">
<meta name="author" content="John Dean">
<title>Halloween on the River</title>
<link rel="stylesheet" href="pumpkinPatch.css">
</head>
```

This `link` element connects the web page to its external CSS file.

**FIGURE 3.9** `head` container for the Pumpkin Patch web page that uses an external CSS file

```
/* John Dean */
.orange {color: darkorange;}
.white {color: white;}
.black {color: black;}
.orange-background {background-color: orange;}
```

CSS comment

**FIGURE 3.10** Source code for external CSS file that gets loaded into the Pumpkin Patch web page

Note the comment syntax. You must start with a `/*` and end with a `*/`.

If you have a long comment, you should have the comment span several lines, like this:

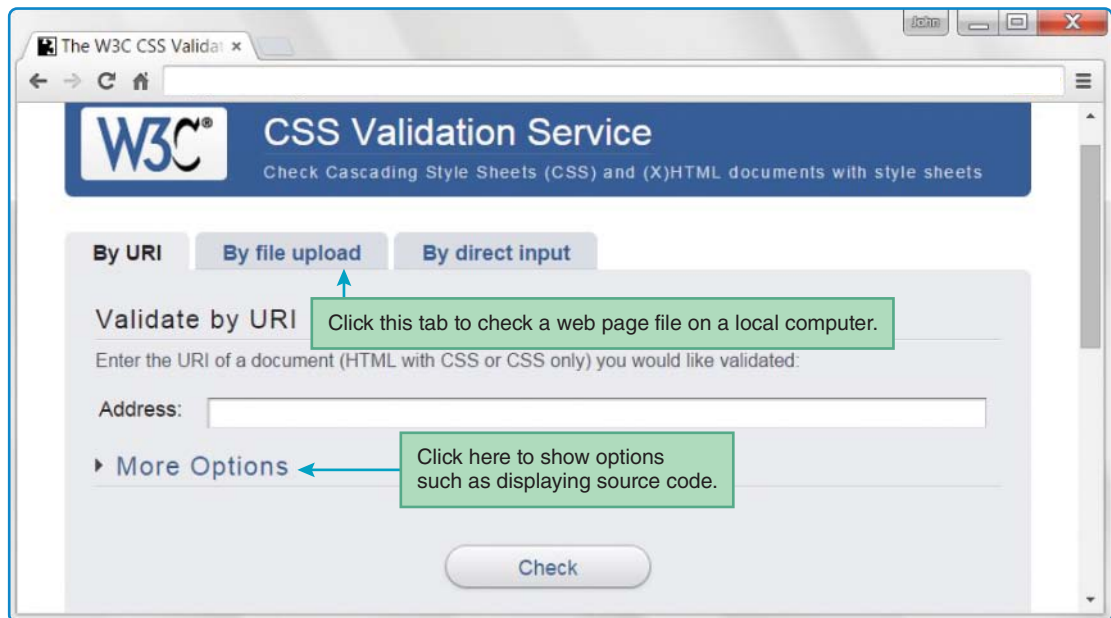
```
/* The following rules are for a CSS image sprite that enables hover
effects for the navigation bar at the left. */
```

This comment mentions a “CSS image sprite.” We’ll describe them in detail later in the book, but for now, just realize that a CSS image sprite is a rather complicated coding construct that requires CSS rules that are nonintuitive. Thus, it provides a good example of something where a CSS comment is appropriate.

## CSS Validation Service

Remember the HTML validation service mentioned in Chapter 1? It’s a great tool for verifying that the code in an HTML file comports with the W3C’s HTML standard. Likewise, there’s a CSS *validation service* tool for verifying that the code in an external CSS file comports with the W3C’s CSS standard. You can find the CSS validation service at <https://jigsaw.w3.org/css-validator>. See **FIGURE 3.11** for a screenshot of the CSS validation service’s home page. We’ll discuss home pages in more depth in a later chapter, but for now, just know that a *home page* is the default first page a user sees when the user visits a website.

In Figure 3.11, note the CSS validation service’s three tabs. With the first tab, **By URI**, the user enters a web address for the external file that is to be checked. For that to work, you need the web



**FIGURE 3.11** W3C’s CSS validation service

page to be uploaded to a web server. With the second option, **By file upload**, the user selects a file on his or her local computer. With the third option, **By direct input**, the user copies HTML code directly into a large text box. Usually, you will use the second option, **By file upload**, because it's a good idea to test a file stored locally before uploading it.

We recommend that you use the CSS validation service to check all your external CSS files. Go ahead and try it out now on the Pumpkin Patch external CSS file. Specifically, retrieve the `pumpkinPatch.css` file from the book's resource center and save it to your hard disk or to a flash drive. Alternatively, you can create the file yourself by loading an IDE, opening a new file, copying Figure 3.10's code into the file, and saving the file with the name `pumpkinPatch.css`. After saving the file, go to the CSS validation service and click the **By file upload** tab. In the **Local CSS file** box, search for and select the `pumpkinPatch.css` file. Click the **Check** button, and that should generate a message indicating success.

## 3.12 CSS Properties

For the remainder of this chapter, we'll focus on CSS properties. As you know from prior examples, a CSS property specifies one aspect of an HTML element's appearance. The W3C's CSS3 standard provides many CSS properties (more than a hundred), so there is great flexibility in terms of specifying appearances. Remembering all those properties and the types of values associated with them can be daunting. Unless you've got the memory of a sea lion,<sup>3</sup> you'll probably need to use a CSS reference and look things up every now and then.

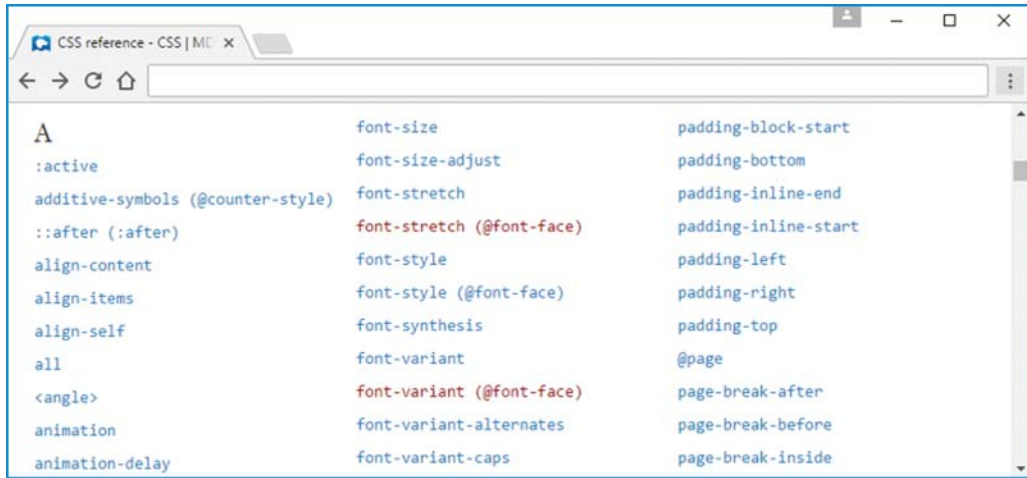
The W3C and the WHATWG have CSS references, but, unfortunately, they're rather disjointed, which can make them difficult to navigate. The Mozilla Developer Network has a more user-friendly CSS reference at <https://developer.mozilla.org/en-US/docs/Web/CSS/Reference>. Take a look at **FIGURE 3.12**, which shows the keyword index part of the reference. Keywords are the words that form the syntax of a language, so the figure's keyword index shows the words that form the CSS language. Note the first keyword entry, `:active`. The keywords that start with a colon are known as pseudo-elements (we'll describe a few of them in later chapters). The non-colonated<sup>4</sup> words are properties. To get details on any of the keywords, click on the keyword. Time for a short field trip: Go to Mozilla's CSS reference and click on the `font-size` property. That takes you to a page with details about the `font-size` property.

We'll discuss quite a few of the CSS properties later in the book, but for this chapter, we'll limit our discussion to the properties shown in **FIGURE 3.13**. Those properties fall into five property groups—color, font, text, border, and margin/padding. We'll discuss the properties in those groups in the upcoming sections.

---

<sup>3</sup>James Randerson, "Sea Lion Scores Top for Memory," *New Scientist*, October 23, 2003, <https://www.newscientist.com/article/dn2960-sea-lion-scores-top-for-memory>.

<sup>4</sup>"Colonated" isn't a word, but it should be. When the *Oxford English Dictionary* folks get around to approving my word submission, colonated will mean "something that has a colon."



**FIGURE 3.12** Mozilla's CSS properties reference

Color properties	<code>color</code> , <code>background-color</code>
Font properties	<code>font-style</code> , <code>font-variant</code> , <code>font-weight</code> , <code>font-size</code> , <code>font-family</code> , <code>font</code>
Text properties	<code>line-height</code> , <code>text-align</code> , <code>text-decoration</code> , <code>text-transform</code> , <code>text-indent</code>
Border properties	<code>border-bottom</code> , <code>border-bottom-color</code> , ...
Margin and padding properties	<code>margin-bottom</code> , <code>margin-left</code> , ... <code>padding-bottom</code> , <code>padding-left</code> , ...

**FIGURE 3.13** CSS properties introduced in this chapter

Note: Ellipses are used here because there are too many border, margin, and padding properties to show in this figure.

## 3.13 Color Properties

In Figure 3.13, you can see two color properties—`color` and `background-color`. The `color` property specifies the color of an element's text. The `background-color` property specifies the background color of an element. The color properties are pretty straightforward, right? It's the *values* for the color properties that require more attention.

There's quite a bit of flexibility when it comes to specifying color values. You can specify a color value using one of five different formats. We'll describe the formats in detail, but first, here's a teaser of what you can look forward to:

- color name—for example, `red`
- RGB value—specifies amounts of red, green, and blue

RGBA value—specifies red, green, and blue, plus amount of opacity

HSL value—specifies amounts of hue, saturation, and lightness


HSLA value—specifies hue, saturation, and lightness, plus amount of opacity

## Color Names

The CSS3 specification defines 147 color names, and the major browsers support all those colors. To view the color names and their associated colors, go to <https://www.w3.org/TR/css3-color/#svg-color>. On that web page, you should recognize a few of the color names, like `orange` and `darkorange`, from previous web page examples in this book. An example of a more obscure color name is `darkslategray`. Note how we use `darkslategray` in this code fragment's class selector rule:

```
<head>
<style>
  .roofColor {color: darkslategray;}
</style>
</head>

<body>
<p>
  Mackay Hall's roof is
  <span class="roofColor">dark slate gray</span>.
</p>
</body>
```



## 3.14 RGB Values for Color

RGB stands for red, green, and blue. An RGB value specifies the amounts of red, green, and blue that mix together to form the displayed color. To specify an amount of a color, you can use a percentage, an integer, or a hexadecimal number (we'll explain hexadecimal shortly). We'll provide explanations and examples coming up, but for now, here are the allowable ranges for each technique:

percentage—0% to 100% for each color

integer—0 to 255 for each color

hexadecimal—00 to ff for each color

### RGB Values with Percentages

To specify an RGB value with percentages, use this format:

```
rgb(red-percent, green-percent, blue-percent)
```

Each percent value must be between 0% and 100%. Here's an example class selector rule that uses an RGB value with percentages:

```
<style>
  .eggplant {background-color: rgb(52%,20%,45%);}
</style>
```

What background color does the preceding rule generate? 20% for the second value means that there is very little green in the color mixture. 52% for the first value means that there is a significant amount of red—52% of the maximum amount of red. 45% for the third value means that there is a bit less blue than there is red. Mixing approximately 50% red, approximately 50% blue, and not much green leads to this dark purple color:



Eggplants are dark purple, and that's why we use `eggplant` in the preceding class selector rule.

If you want to specify black, then you need to use the least intensity (a value of 0%) for each of the three colors. That should make sense because, as any physics major will tell you, black is the absence of all light. Therefore, here's the CSS rule for a black background:

```
.black {background-color: rgb(0%,0%,0%);}
```

To specify white, you need to use the greatest intensity (a value of 100%) for each of the three colors. That should make sense because white is the combination of all colors.<sup>5</sup> Therefore, here's the CSS rule for white text (which would work nicely with the preceding black background):

```
.white {color: rgb(100%,100%,100%);}
```

## RGB Values with Integers

To specify an RGB value with integers, use this format:

```
rgb(red-integer, green-integer, blue-integer)
```

Each integer value must be between 0 and 255, with 0 providing the least intensity and 255 providing the most. Here are two class selector rules that use RGB values with integers:

```
<style>
  .favorite1 {color: rgb(144,238,144);}
  .favorite2 {color: rgb(127,127,127);}
</style>
```

What text color does the `favorite1` class selector rule produce? The larger value, 238, is for green, so it produces a shade of green. The red and blue color values are 144, which is more than halfway to the maximum value of 255. That means the green color will be closer to white than black. The result is a light green. By the way, if you go to the W3C's CSS color names web page, you can find a `lightgreen` color name, and you can see that it matches the preceding `rgb(144,238,144)` value.

What text color does the `favorite2` class selector rule produce? All three color values (127, 127, 127) are the same, so it produces a color somewhere between white and black. Since 127 is halfway to the maximum value of 255, `favorite2` produces a medium gray color.

---

<sup>5</sup> In 1666, Isaac Newton discovered that white light is composed of all the colors in the color spectrum. He showed that when white light passes through a triangular prism, it separates into different colors. And when the resulting colors pass through a second triangular prism, they are recombined to form the original white light.

## RGB Values with Hexadecimal

With many programming languages, including HTML and CSS, numbers can be represented not only with base-10 decimal numbers, but also with base-16 hexadecimal (hex) numbers. A number system’s “base” indicates the number of unique symbols in the number system. So base 10 (for the decimal number system) means there are 10 unique symbols—0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. And base 16 (for the hexadecimal number system) means there are 16 unique symbols—0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F.

When specifying an RGB color value, you have choices. You can use percentages or standard integers as described earlier, or you can use hexadecimal values. For hexadecimal RGB values, you’ll need to use the format `#rrggbb` where:

`rr` = two hexadecimal digits that specify the amount of red  
`gg` = two hexadecimal digits that specify the amount of green  
`bb` = two hexadecimal digits that specify the amount of blue

With the percentage and integer RGB values, a smaller number for a particular color means less of that color. Likewise, with hexadecimal RGB values, a smaller number means less of a particular color. The smallest hexadecimal digit is 0, so `00` represents the absence of a particular color. If all colors are absent, that’s black. Therefore, `#000000` (`00` for each of the three colors) indicates black. The largest hexadecimal digit is `f`, so `ff` represents the greatest intensity of a particular color. If all colors are maximally intense, that’s white. Therefore, `#ffffff` (`ff` for each of the three colors) indicates white.

So you now know how to make black and white using hexadecimal numbers. Those are easy. Using hexadecimal numbers that are between the two extremes is a bit trickier. What color does the following value represent?

```
#ffbbbb
```

Note `ff` for the first two digits. The first two digits are for red, so `#ffbbbb` is a shade of red. Light red or dark red? To figure that out, you need to know whether the other two colors are less than or greater than halfway between `00` and `ff`. The other two colors both have values of `bb`. The `b` hex digit (which represents eleven) is closer to `f` (which represents fifteen) than `0`, so the other two colors are closer to white than black. That means `#ffbbbb` is light red (otherwise known as pink), and it looks like this:



Here’s an example CSS rule with a hexadecimal RGB value:

```
<style>
  .sapphire {background-color: #0f42ba;}
</style>
```

In the `#0f42ba` color value, which is the most intense—red, green, or blue? The red portion’s value, `0f`, has an `f` in it, and we know `f` represents fifteen, so maybe red is the most intense? No. With good old-fashioned base-10 decimal numbers, `09` is a small number (as compared to, say, `95`) because digits at the left (`0` in this case) have greater significance than digits at the right (`9` in this case).



It's the same in the hexadecimal number system. The red portion of #0f42ba is 0f, and now we know that 0f is a small number; consequently, there's very little red in the resulting color. The green portion of #0f42ba is 42, and 42 is a relatively small number; consequently, there's not much green in the resulting color. The blue portion of #0f42ba is ba, and ba is a relatively large number;<sup>6</sup> consequently, there's quite a bit of blue in the resulting color. Here's what the color looks like:



That is the color of a sapphire gemstone, and that's why the preceding CSS rule uses `sapphire` for its class selector.

The W3C's CSS3 standard says that RGB hexadecimal digits are case insensitive. So in this example, #0F42BA (with uppercase F, B, and A) would also work. You'll see lowercase and uppercase on the W3C site and in the real world. Use lowercase to be consistent with standard HTML coding conventions, which say to use lowercase values unless there's a reason to do otherwise. But if you're reading this book as part of a course, and your teacher says to use uppercase hexadecimal digits, then use uppercase hexadecimal digits.

## 3.15 Opacity Values for Color

In the previous section, you learned how to produce a color by using the `rgb` construct with values for red, green, and blue. In this section, you'll learn how to produce a partially transparent color by using the `rgba` construct with values for red, green, blue, and opacity. The fourth value, opacity, determines how opaque the color is, where opaque refers to the inability to see through something. It's the opposite of transparency. If the opacity value is 100%, that means the color is completely opaque, and if there is content behind the color, that content gets covered up. At the other extreme, if the opacity value is 0%, that means the color is completely transparent.

The A in RGBA stands for alpha. Why alpha? There's no official documentation on alpha's etymology, but perhaps the CSS3 standards people chose it simply because it's the first letter in the Greek alphabet.

To specify an RGBA value, use one of these two formats:

```
rgba (red-integer , green-integer , blue-integer , opacity-number-between-0-and-1)  
rgba (red-percent , green-percent , blue-percent , opacity-number-between-0-and-1)
```

For both formats, the fourth value specifies the opacity. The opacity value must be in the form of a decimal number between 0 and 1, with 0 being completely transparent, 1 being completely opaque, and .5 in between.

For the first format, each integer value must be between 0 and 255, with 0 providing the least intensity and 255 providing the most. That should sound familiar because that was also the case for integers with the `rgb` construct. For the second format, each percent value must be between 0% and 100%.

---

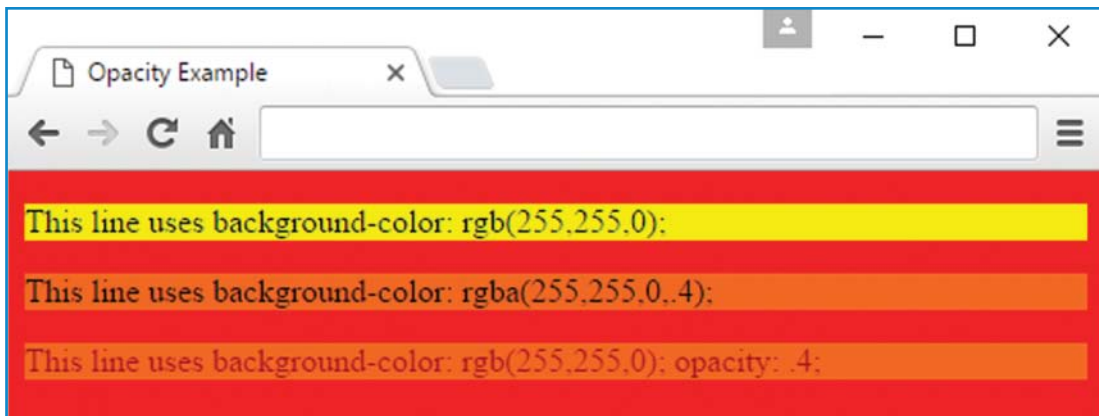
<sup>6</sup> To calculate the decimal equivalent of a two-digit hexadecimal number, multiply the left digit by 16 and then add the right digit. So, hexadecimal 0f is 15 ( $0 \times 16 + 15 = 15$ ), hexadecimal 42 is 66 ( $4 \times 16 + 2 = 66$ ), and hexadecimal ba is 186 ( $11 \times 16 + 10 = 186$ ).

You can use the `rgba` construct with the `color` property, which specifies the color of an element's foreground (e.g., text). However, it's more common to use it with the `background-color` property, as you will see in the following example.

**FIGURE 3.14's** Opacity Example web page illustrates what happens when a transparent yellow color is placed on top of a red background—orange is formed. Before looking at the CSS rules that generate the yellow colors, let's first examine the CSS rule that generates the window's red background color:

```
.red {background-color: red;}
```

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<meta name="author" content="John Dean">
<title>Opacity Example</title>
<style>
  .red {background-color: red;}
  .yellow-bg {background-color: rgb(255,255,0);}
  .yellow-bg2 {background-color: rgba(255,255,0,.4);}
  .yellow-bg3 {background-color: rgb(255,255,0); opacity: .4;}
</style>
</head>
<body class="red">
<p class="yellow-bg">This line uses background-color: rgb(255,255,0);</p>
<p class="yellow-bg2">
  This line uses background-color: rgba(255,255,0,.4);
</p>
<p class="yellow-bg3">
  This line uses background-color: rgb(255,255,0); opacity: .4;
</p>
</body>
</html>
```



**FIGURE 3.14** An opacity example that illustrates the `rgba` construct and the `opacity` property

Note the browser window's first sentence and the CSS rule that generates its yellow background:

```
.yellow-bg {background-color: rgb(255,255,0);}
```

The `rgb` value's 255 values indicate maximum amounts of red and green, and that forms yellow. The web page's red background is blocked by the sentence's yellow background, which is 100% opaque by default.

Next, note the browser window's second sentence and the CSS rule that generates its orange background:

```
.yellow-bg2 {background-color: rgba(255,255,0,.4);}
```

The `.4` indicates that the yellow color will be 40% opaque and 60% transparent. That transparency allows the yellow background to mix with the web page's red background to form orange.

As an alternative to using `rgba`, you can use `rgb` in conjunction with the `opacity` property. Here's the third class selector rule from the Opacity Example web page:

```
.yellow-bg3 {background-color: rgb(255,255,0); opacity: .4;}
```

This rule matches up with the web page's third sentence. With an opacity value of `.4`, that sentence gets 40% opacity for both its `color` property and its `background-color` property. For the sentence's `color` property, which determines the text's color, the default color is black, and the web page's background color is red. They mix to form a reddish gray. Note the reddish-gray text for the Opacity Example web page's third sentence. For the sentence's `background-color` property, the preceding CSS rule specifies yellow, and the web page's background color is red. They mix to form orange. Note the orange background color for the Opacity Example web page's third sentence.

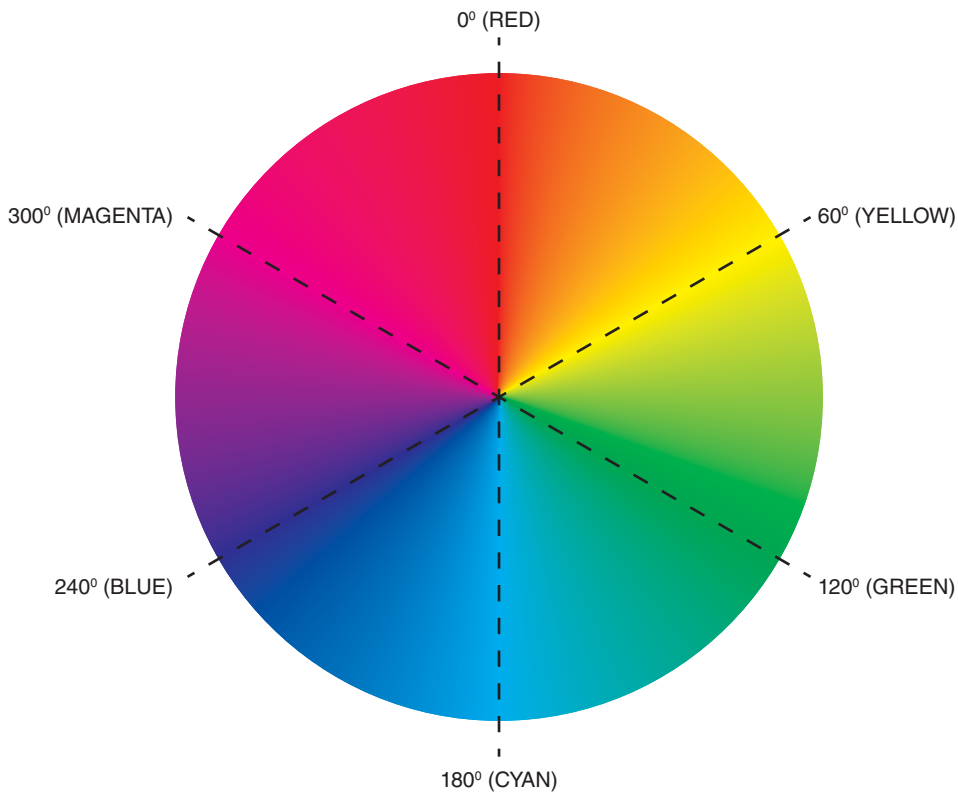
## 3.16 HSL and HSLA Values for Color

With CSS (and also with HTML), there are sometimes multiple ways to accomplish the same thing. That point is particularly pertinent when it comes to specifying colors. You've already learned how to specify color values with names, the `rgb` construct, and the `rgba` construct. Now for the `hsl` construct. Here's the syntax:

```
hsl (hue-integer, saturation-percent, lightness-percent)
```

HSL stands for hue, saturation, and lightness. Hue is a degree on the color wheel shown in **FIGURE 3.15**. The wheel is a circle, so the wheel's degrees go from 0 to 360. As you can see in the figure, 0 degrees is for red, 120 degrees is for green, and 240 degrees is for blue. For a circle, 0 degrees is equivalent to 360 degrees. So, to specify red, you can use 360 as an alternative to 0.

The second value in the `hsl` construct is the color's percentage of saturation. The W3C says 0% means a shade of gray, and 100% is the full color. For an alternative way to think about it, if you look up "saturation" on Google, you should find something like "the extent to which something is dissolved or absorbed compared with the maximum possible absorption." Suppose there's a girl



**FIGURE 3.15** Color wheel for the `hsl` construct's hue value

named Caiden on her middle school cheer team. She washes her new \$50 uniform and forgets to remove her cherry nail polish from the uniform's skirt pocket. The result? (1) A big blotch of "maximum possible absorption" red near the pocket due to a saturation value of 100%, (2) a bit of reddish coloring added to the rest of the uniform due to a saturation value near 25%, and (3) a month's worth of Caiden doing the whole family's laundry, nail polish excluded.

The third value in the `hsl` construct is the color's percentage of lightness. A lightness value of 0% generates black, regardless of the values for hue and saturation. A lightness value of 100% generates white, regardless of the values for hue and saturation. A lightness value of 50% generates a "normal" color.

Here's an example CSS rule with an HSL color value:

```
<style>
  p {background-color: hsl(120,50%,75%);}
</style>
```

That color forms a light shade of grayish green, and here's what it looks like:



In the previous section, you learned how to add transparency to an RGB value by using the `rgba` construct. Likewise, to add transparency to an HSL value, you can use the `hsla` construct. Here's the syntax:

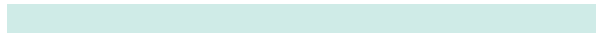
```
hsla (hue-integer , saturation-percent , lightness-percent , opacity-number-between-0-and-1)
```

The fourth argument specifies the opacity. The opacity value must be in the form of a decimal number between 0 and 1, with 0 being completely transparent and 1 being completely opaque.

Here's an example CSS rule with an HSLA color value:

```
<style>
  .background {background-color: hsla(120,50%,75%,.5);}
</style>
```

What color is specified? It's the same as the earlier grayish-green color, except that this time, the grayish green blends with the web page's background color as a result of the 50% opacity value. With a default white web page background, the result would be a lighter shade of grayish green, like this:



## 3.17 Font Properties

In this section, we describe how to display text with different font characteristics. You've probably heard of the term "font," but if not, *font* refers to the characteristics of text characters—height, width, thickness, slantedness,<sup>7</sup> body curvatures, and endpoint decorations. That should make more sense later on when we present CSS font property details and show examples. Specifically, you'll learn about the `font-style`, `font-variant`, `font-weight`, `font-size`, `font-family`, and `font` shorthand properties.

### font-style Property

The `font-style` property specifies whether the text is to be displayed normally or slanted. Here are the valid values for the `font-style` property:

font-style Values	Description
normal	Upright characters (not slanted).
oblique	Use the same font as the current font, but slant the characters.
italic	Use a cursive font (which tends to be slanted and is supposed to look like handwriting).

<sup>7</sup> Apparently "slantedness" is not a word, but it should be. If it becomes a word, you heard it here first.

These descriptions indicate a slight difference between the `oblique` and `italic` properties, with `italic` tending to be more decorative. Most web developers use the value `italic`. Because italics are so common, you should memorize the following technique for generating italics:

```
.italics {font-style: italic;}
```

As always, choose a name for the class selector that's descriptive. Here, we chose the name `italics` because it's descriptive and easy.

Upright (normal) characters are the default, so why would you ever want to specify normal for the `font-style` property? Suppose you have a whole paragraph that's italicized and you want one word in the paragraph not italicized. To make that word normal (not italicized), you can use `font-style: normal`.

The W3C provides default values for all CSS properties, and to force the default value to be used for a particular property, you can specify `initial` for that property. So, given the situation described with an italicized paragraph, to make one word not italicized, you can apply the following rule to that word:

```
.not-italicized {font-style: initial;}
```

## font-variant Property

The `font-variant` property specifies how lowercase letters are displayed. Here are the valid values for the `font-variant` property:

font-variant Values	Description
<code>normal</code>	Display lowercase letters normally.
<code>small-caps</code>	Display lowercase letters with smaller-font uppercase letters.

Here's an example that uses a `small-caps` CSS rule:

```
.title {font-variant: small-caps;}
...
<div class="title">The Great Gatsby</div>
```

And here's the resulting displayed text:

THE GREAT GATSBY

## font-weight Property

The `font-weight` property specifies the boldness of the text characters. Here are the valid values for the `font-weight` property:

font-weight Values	Description
normal, bold	It's up to the browser to determine a font weight that can be described as normal or bold.
bolder, lighter	Using a value of <code>bolder</code> causes its targeted text to have thicker characters than the text that surrounds it. Using a value of <code>lighter</code> causes its targeted text to have thinner characters than the text that surrounds it.
100, 200, 300, 400, 500, 600, 700, 800, 900	100 generates the thinnest characters and 900 the thickest characters. 400 is the same as <code>normal</code> , and 700 is the same as <code>bold</code> .

These descriptions for `bolder` and `lighter` are probably all you need to know, but you may want to dig a little deeper. With `bolder` and `lighter`, the targeted text inherits a default `font-weight` value from its surrounding text, and then the targeted text's weight gets adjusted up or down relative to that inherited weight. For example, if you specify a `bold` font weight for a paragraph, you can make a particular word within the paragraph even `bolder` by specifying `bolder` for that word's font weight.

Because boldfacing is such a common need, you should memorize the following technique for making something bold:

```
.bold {font-weight: bold;}
```

## font-size Property

The `font-size` property specifies the size of the text characters. There are quite a few values allowed for the `font-size` property. Here are the most appropriate ones:

Here's an example class selector rule that uses the `font-size` property with an `xx-large` value:

font-size Values	Description
xx-small, x-small, small, medium, large, x-large, xx-large	It's up to the browser to determine a font size that can be reasonably described as <code>xx-small</code> , <code>x-small</code> , <code>small</code> , etc.
smaller, larger	Using a value of <code>smaller</code> causes its targeted text to have smaller characters than the text that surrounds it. Using a value of <code>larger</code> causes its targeted text to have larger characters than the text that surrounds it.
number of em units	One em unit is the height of the element's normal font size.

```
.huge-font {font-size: xx-large;}
```

So far, most of the CSS property values you've seen have consisted of a text description, such as `xx-large`. As an alternative, some properties have values that are comprised of two parts—a number and a unit. For example, for the `font-size` property, you can use a value with a number next to `em`, where one `em` unit is the height of a typical character. The `em` unit's name comes from the letter M. Originally, one `em` unit equaled the height of the letter M. However, there are fonts for languages that don't use the English alphabet, so it was deemed inappropriate for `em` to rely on the letter M. Thus, `em` is no longer tied to the letter M, and testing shows that a single `em` unit is a bit taller than the height of M.

Here are class selector rules that use the `font-size` property with `em` values:

```
.disclaimer {font-size: .5em;}  
.advertisement {font-size: 3em;}
```

The first rule is for disclaimer text, which is supposed to be annoyingly small to avoid scrutiny, and its `.5em` value displays text that is half the size of normal text. The second rule is for advertisement text, which is supposed to be annoyingly large to draw attention, and its `3em` value displays text that is three times the size of normal text.

Here, for each `font-size` value, note that there is no blank space separating the number from `em`. Likewise, for all CSS values that consist of a number and a unit, you should not have a blank space. Having no blank space is a requirement of the CSS standard.

## Absolute Units

There are quite a few other techniques for specifying font size. For example, you can specify a number along with an `in`, `cm`, `mm`, `pt` (point), or `pc` (pica) unit. The W3C refers to those units rather disdainfully as “so-called absolute units.” Although they're still used every now and then, and you should understand them, absolute units have fallen out of favor for everyday needs. This is because it's good to allow users to adjust the size of things, particularly people with impaired eyesight. Also, with regular monitors, testing shows that the “absolute units” are not absolute; their sizes vary with different resolutions, different monitors, and zooming in and out.

The W3C says absolute units are required to have absolute lengths only when the target device has a high resolution; that usually means just printers, but it can also mean high-resolution monitors. For a scenario where a fixed size is essential and absolute units are appropriate, picture your wedding preparations. Suppose you're tasked with printing your wedding invitations on pre-cut pink cherub-adorned cards. To avoid agitating a stressed-out fiancé(e), you'll want to use absolute units to make sure the words fits perfectly on the cards.

## font-family Property

The prior font properties allow the web developer to choose values for specific font characteristics (`font-weight` for characters' thickness, `font-size` for characters' height, etc.). The next font



property, `font-family`, is more holistic in nature. The `font-family` property allows the web developer to choose the set of characters that the browser uses when displaying the element's text. As you'd expect, the characters in a particular character set are similar in appearance—same basic height, width, body curvature, and so on.

Here's an example class selector rule that uses the `font-family` property:

```
.ascii-art {font-family: Courier, Prestige, monospace;}
```

Note that with the `font-family` property, you should normally have a comma-separated list of fonts, not just one font. In applying the preceding rule to elements that use “`ascii-art`” for their `class` attribute, the browser works its way through the `font-family` list from left to right, and uses the first font value it finds installed on the browser's computer and skips the other fonts. So if `Courier` and `Prestige` are both installed on a computer, the browser uses the `Courier` font because it appears further left in the list.

There are lots of fonts, and different browsers support different ones. Users are able to install fonts in addition to the ones provided by their browsers. For a small sample of popular font names, see <https://www.w3.org/TR/css-fonts-3/#generic-font-families>. On that website, you can see that most font names use title case (the first letter of each word is capitalized), but a few use all lowercase letters. Font names are case sensitive on some operating systems, so you should take care to use the proper case. The font names that use all lowercase letters are special, and they are known as generic fonts.

A *generic font* is a name that represents a group of fonts that are similar in appearance. For example, `monospace` is a generic font, and it represents all the fonts where each character's width is uniform. Whenever you use a `font-family` CSS rule, you should include a generic font at the end of the rule's list of font names. In the following CSS rule (copied from earlier for your convenience), note the `monospace` font at the end of the list of font names:

```
.ascii-art {font-family: Courier, Prestige, monospace;}
```

The generic font name provides a *fallback mechanism*. If a user's browser doesn't support any of the fonts at the left of the generic font name, the browser uses the generic font to display a font that the browser does support. Generic font names are not actual fonts with specific appearance characteristics; they are just placeholders that tell the browser to look for an actual font in a particular family of fonts. So in the `ascii-art` CSS rule, if a browser does not support the `Courier` or `Prestige` fonts, the browser digs up a `monospace` font that it does support and uses it. To ensure that a web page's font looks pretty much the same on different browsers, for each `font-family` CSS rule, you should always use fonts that are similar, which means that they are associated with the same generic font. Referring once again to the preceding `ascii-art` CSS rule, `Courier` and `Prestige` both display uniform-width characters, and they are both associated with the same `monospace` generic font.

The `monospace` font is one of five generic fonts that all browsers recognize and support. Here's a list of the five generic fonts with descriptions and examples:

Generic Font Names	Description	Example Font
monospace	All characters have the same width.	Courier New looks like this.
serif	Characters have decorative embellishments on their endpoints.	Times New Roman looks like this.
sans-serif	Characters do not have decorative embellishments on their endpoints.	Arial looks like this.
cursive	Supposed to mimic cursive handwriting, such that the characters are partially or completely connected.	<i>Monotype Corsiva looks like this.</i>
fantasy	Supposed to be decorative and playful.	<b>Impact looks like this.</b>

As mentioned earlier, generic font names use lowercase, and you can verify that for the five generic fonts shown. On the other hand, specific font names use title case. You can verify that by examining the five example fonts shown in the right column (e.g., *Courier New*).

When specifying a multiple-word font name, surround the name with quotes. For example, in **FIGURE 3.16**, note the quotes around *New Century Schoolbook*. In addition, note the coding convention of inserting a blank space after each comma in a `font-family` list of font names.

Suppose a web page includes the code in Figure 3.16. If a user's computer has the *New Century Schoolbook* font and also the *Times* font installed on it, which font will the browser use to display the paragraph? It will use *New Century Schoolbook*, because it's the first font in the list and it's installed on the user's computer.

```

<style>
  blockquote {font-family: "New Century Schoolbook", Times, serif;}
</style>

<blockquote>
  Call me Ishmael. Some years ago-never mind how long precisely-
  having little or no money in my purse, and nothing particular to
  interest me on shore, I thought I would sail about a little and
  see the watery part of the world.
</blockquote>
<cite>Moby Dick</cite>

```

The diagram shows two green boxes labeled 'quotes' and 'spaces'. An arrow points from the 'quotes' box to the opening and closing double quotes around 'New Century Schoolbook' in the CSS code. Another arrow points from the 'spaces' box to the blank spaces after the commas in the font-family list: 'New Century Schoolbook', Times, serif.

**FIGURE 3.16** An example font-family CSS rule

## font Shorthand Property

Fairly often as a web programmer, you'll want to apply more than one of the prior font-related properties to an element. You could specify each of those font properties separately, but there's an easier way. The `font` property can be used to specify all these more granular font properties—`font-style`, `font-variant`, `font-weight`, `font-size`, `line-height`, and `font-family`. Previously, we mentioned all these font properties except for `line-height`. We'll cover `line-height` shortly, but first we'll discuss some overarching details about the `font` property.


Here's the syntax for a `font` property-value pair:

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


As usual, the italics are the book's way of telling you that the italicized thing is a description of what goes there, so for a `font` property-value pair in a CSS rule, you would replace *font-style-value* with one of the `font-style` values, such as `italic`. The square brackets are the book's way of telling you that the bracketed thing is optional, so the `font-style`, `font-variant`, `font-weight`, and `line-height` values are all optional. On the other hand, the `font-size` and `font-family` values have no square brackets, so you must include them whenever you use the `font` property. For the property values you decide to include, they must appear in the order previously shown. If a `line-height` value is included, you must position it at the right of the `font-size` value, with a `/` separating the two values.

Here's an example type selector rule that uses a `font` property:

```
blockquote {
  font: italic large "Arial Black", Helvetica, sans-serif;
}
```



Use spaces to separate property values.



Use commas to separate font-family values.

In the preceding rule, how many types of font properties are specified and what are they? Glancing at the `font` property's value, you can see there are five items in the list, so you might think there are five types of font properties. Upon closer inspection, you should notice the commas between the last three items. Those commas are delimiters for a sublist, where the sublist is the value for the `font-family` property. At the left of the `font-family` property's list, you can see two other property values—`italic` and `large`—separated by spaces. Those values are for the `font-style` and `font-size` properties.

By the way, the W3C refers to the `font` property as a *shorthand property* because it's a time-saving construct for handling multiple font characteristics. Later, we'll introduce additional shorthand properties for other groups of related CSS properties. When given a choice between using a shorthand property and using a set of more granular properties, some web programmers prefer using the shorthand property because of its compactness, whereas others prefer using the more

granular properties because of their clarity. In this book, we use both techniques and let the situation dictate our preference.