3.9 Cascading

Have you wondered about the significance of the words in "Cascading Style Sheets"? Traditionally, a "style sheet" is a collection of rules that assign appearance properties to structural elements in a document. For a web page, a style sheet "rule" refers to a value assigned to a particular display property of a particular HTML element. The "cascading" part of Cascading Style Sheets is the subject of this section. If you look up the word "cascade" online or in a dictionary,[2] you'll see something like "a series of stages in a process." Likewise, CSS uses a series of stages. More specifically, there are different stages/places where CSS rules can be defined. Each stage/place has its own set of rules, and each set of rules is referred to as a style sheet. With multiple style sheets organized in a staged structure, together it's referred to as Cascading Style Sheets.

To handle the possibility of conflicting rules at different places, different priorities are given to the different places. See **FIGURE 3.8**, which shows the places where CSS rules can be defined. The higher priority places are at the top, so an element's `style` attribute (shown at the top of the CSS rules list) has the highest priority. We'll explain the `style` attribute in the next section, but let's first do a cursory run-through of the other items in Figure 3.8's list.

Figure 3.8 shows that the second place for CSS rules is in a `style` container. That placement should sound familiar because we've been using `style` containers for all the prior CSS examples. The next place for CSS rules is in an external file. We'll discuss external files later in this chapter.

The next place for CSS rules is in the settings defined by a user for a particular browser installation. We won't discuss that technique because there's nothing for you, the programmer, to learn or to do. Instead, the user may choose the settings he or she is interested in. If you'd like to learn how to choose the settings, you're on your own, and be aware that different browsers have different interfaces for adjusting their settings.

The last place for CSS rules, and the place with the lowest priority, is in the native default settings for the browser that's being used. As a programmer, there's nothing you can do to modify a browser's native default settings. But you should be aware of those default settings so you know what to expect when none of the first four cascading techniques is employed. In Chapter 2, we described "typical default display properties" for each HTML element presented. Those properties come from the major browsers' native default settings. Different browsers can have different default settings, so your web pages might not look exactly the same on different browsers. In general, try not to rely on browser defaults because it's hard to gauge the whims of the browser gods as they churn out new browser versions (with possibly new browser defaults) at a precipitous pace.

In displaying an element, a browser will check for CSS rules that match the element, starting the search at the top of the cascading CSS rules list in Figure 3.8 and continuing the search

[2] A *dictionary* was an ancient form of communication, used as a means to record word definitions. The definitions appeared on thin sheets of compressed wood fiber.

Places Where CSS Rules Can Be Defined, Highest to Lowest Priority
1. In an element's `style` attribute.
2. In a `style` element in the web page's head section.
3. In an external file.
4. In the settings defined by a user for a particular browser installation.
5. In the browser's native default settings.

FIGURE 3.8 Places where CSS rules can be defined

down the list, as necessary. When there is a CSS rule match, the CSS rule's properties will be applied to the element, and the search down the list stops for those properties.

3.10 `style` Attribute, `style` Container

`style` Attribute

The `style` attribute is at the top of Figure 3.8's cascading CSS rules list; as such, when you use the `style` attribute for CSS rules, those rules are given the highest priority. Here's an example element that uses a `style` attribute:

```
<h2 style="text-decoration:underline;">Welcome!</h2>
```

As you can see, using the `style` attribute lets you insert CSS property-value pairs directly in the code for an individual element. So the preceding `h2` element—but no other `h2` elements—would be rendered with an underline.

The `style` attribute is a *global attribute*, which means it can be used with any element. Even though it's legal to use it with every element, and you'll see it used in lots of legacy code, you should avoid using it in your pages. Why? Because it defeats the purpose of CSS—keeping presentation separate from content.

Let's imagine a scenario that demonstrates why the `style` attribute is bad. Suppose you embed a `style` attribute in each of your `p` elements so they display their first lines with an indentation (later on, you'll learn how to do that with the `text-indent` property). If you want to change the indentation width, you'd have to edit every `p` element. On the other hand, making such a change is much easier when the CSS code is at the top of the page in the `head` container because you only have to make the change in one place—in the `p` element's class selector rule. If you make the change there, it affects the entire web page.

Using the `style` attribute used to be referred to as "inline styles," but the W3C no longer uses that term, and we won't use it either. However, you should recognize the term "inline styles" because you'll hear it being used every now and then.

`style` Container

As you know from prior examples, the `style` element is a container for CSS rules that apply to the entire current web page. The browser applies the CSS rules' property values by matching the CSS rules' selectors with elements in the web page. Normally, you should have just one `style` container per page, and you should put it in a web page's `head` container. It's legal to put a `style` container in the `body`, but don't do it because then it's harder to find the CSS rules.

In Figure 3.8's cascading CSS rules list, note how the higher priority places are more specific. More specific rules beat more general rules. For example, if a `style` attribute designates a paragraph as blue, but a rule in a `style` container designates paragraphs as red, then what color will the browser use to render the paragraph? The `style` attribute's blue color wins, and the browser renders that particular paragraph with blue text. This principle of more specific rules beating more general rules should sound familiar. It parallels the principle introduced earlier that says local things override global things.

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.

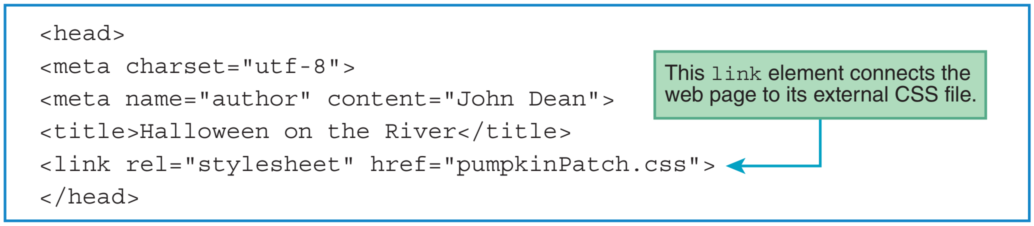
Example

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:

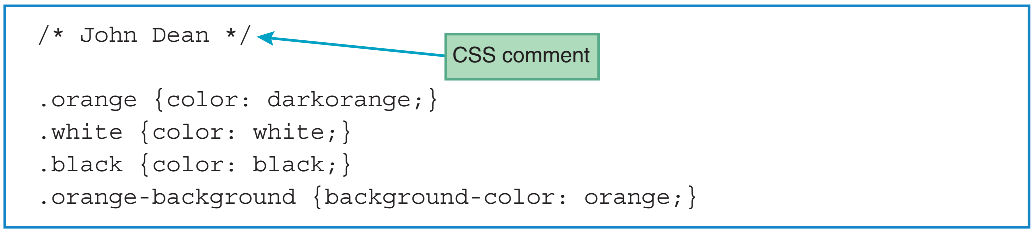
```
/* John Dean */
```



```
<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:

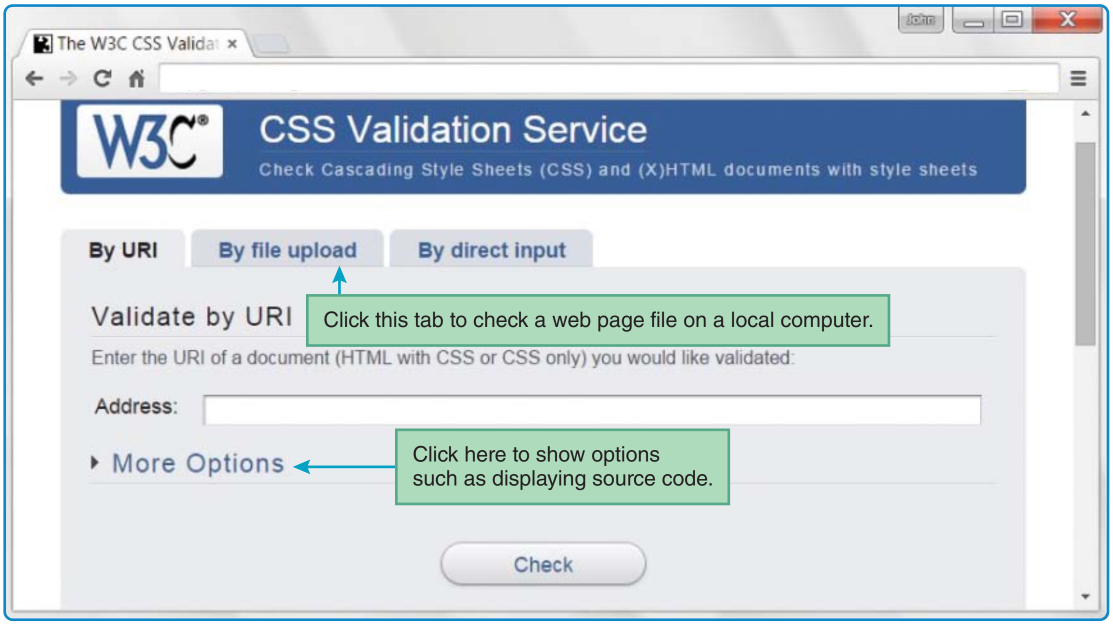
```
/* The following rules are for a CSS image sprite that enables hover
effects for the navigation bar at the left. */
```

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


CSS Validation Service

Check Cascading Style Sheets (CSS) and (X)HTML documents with style sheets

Validate by URI

Enter the URI of a document (HTML with CSS or CSS only) you would like validated:

Address:

Click this tab to check a web page file on a local computer.

Click here to show options such as displaying source code.

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

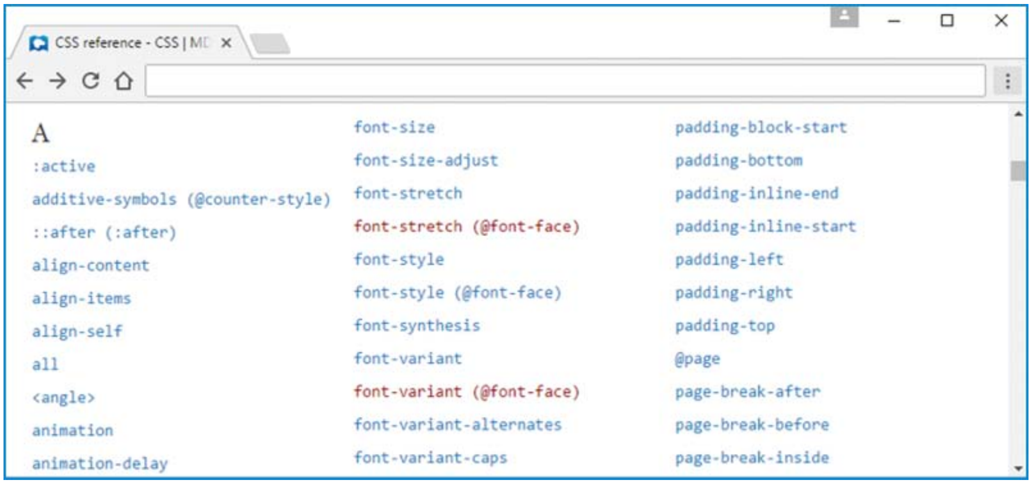
For the remainder of this chapter, we'll focus on CSS properties. As you know from prior examples, a CSS property specifies one aspect of an HTML element's appearance. The W3C's CSS3 standard provides many CSS properties (more than a hundred), so there is great flexibility in terms of specifying appearances. 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,[3] 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[4] 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.

[3] 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.

[4] "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."



A

:active
additive-symbols (@counter-style)
::after (:after)
align-content
align-items
align-self
all
<angle>
animation
animation-delay

font-size
font-size-adjust
font-stretch
font-stretch (@font-face)
font-style
font-style (@font-face)
font-synthesis
font-variant
font-variant (@font-face)
font-variant-alternates
font-variant-caps

padding-block-start
padding-bottom
padding-inline-end
padding-inline-start
padding-left
padding-right
padding-top
@page
page-break-after
page-break-before
page-break-inside

FIGURE 3.12 Mozilla's CSS properties reference

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

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>
                         color name

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