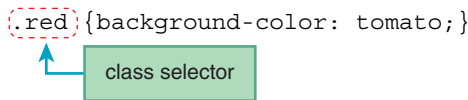


3.6 Class Selectors

Class Selector Overview

So far, we've talked about type selectors and the universal selector. We're now going to talk about a third type of CSS selector—a *class selector*. Let's jump right into an example. Here's a class selector rule with `.red` for its class selector and a background tomato color for matched elements:

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



The dot thing (`.red` in this example) is called a class selector because its purpose is to select elements that have a particular value for their class attribute. So the class selector rule would select/match the following element because it has a `class` attribute with a value of `red`:

```
<q class="red">It is better to keep your mouth closed and let people think you are a fool than to open it and remove all doubt.</q>
```

In applying the class selector rule to this element, the quote gets displayed with a tomato background color.

As with type selectors, you can have more than one class selector share one CSS rule. Just separate the selectors with commas and spaces, like this:

```
.red, .conspicuous, h1 {background-color: tomato;}
```

Note that in addition to a second class selector (`.conspicuous`), there's also a type selector (`h1`). In a single CSS rule, you can have as many comma-separated selectors as you like, all sharing the same set of property-value pairs.

With a type selector, your selector name (`h1` in this example) comes from the set of predefined HTML element names. But for a class selector, you make up the selector name. When you make up the selector name, make it descriptive, as is the case for `red` and `conspicuous` in the preceding example. As an alternative for `red`, you could get even more descriptive and use `tomato`. If you use `tomato`, that will be the same as the name used by the property value. There isn't anything wrong with that. Consistency is good.

Now let's look at class selectors in the context of a complete web page. In **FIGURE 3.3**, note the three CSS rules with their class selectors `.red`, `.white`, and `.blue`. Then take a look at the three `q` elements and their class attribute clauses `class="red"`, `class="white"`, and `class="blue"`. Try to figure out what the web page will display before moving on to the next paragraph.

In Figure 3.3, the first `q` element has a class attribute value of `red`, which means the `.red` CSS rule applies. That causes the browser to display the first `q` element with a tomato-colored

```

<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<meta name="author" content="John Dean">
<title>Mark Twain Quotes</title>
<style>
  .red {background-color: tomato;}
  .white {background-color: white;}
  .blue {background-color: skyblue;}
  q {font-family: Impact;}
</style>
</head>

<body>
<h1>Mark Twain Quotes</h1>
<q class="red">It is better to keep your mouth closed and
  let people think you are a fool than to open it and
  remove all doubt.</q><br>
<q class="white">Get your facts first, then you can distort
  them as you please.</q><br>
<q class="blue">Never put off till tomorrow what you can do
  the day after tomorrow.</q>
</body>
</html>

```

FIGURE 3.3 Source code for Mark Twain Quotes web page

background. I used a standard red background initially, but I found that the black text didn't show up very well. Thus, I chose tomato red, since it's lighter, and the color reminds me of my cherished home-grown tomatoes. Moral of the story: Get used to trying things out, viewing the result, and changing your code if appropriate.

The second and third `q` elements have `class` attribute values of `white` and `blue`. As you can see from the source code, that means they get matched with the `.white` and `.blue` class selector rules, and they get rendered with `white` and `skyblue` backgrounds, respectively. Take a look at **FIGURE 3.4** and note the red, white, and blue background colors for the three quotes.

In addition to the three class selector rules, the Mark Twain Quotes web page also has a type selector rule, `q {font-family: Impact;}`. We'll discuss the `font-family` property later in this chapter, but for now, look at the Mark Twain quotes web page and observe the thick block lettering for the three `q` elements. That lettering is from the `Impact` font.

Usually, browsers use a default background color of white, so why did we specify `white` for the second `q` element's background color? One benefit is that it's a form of self-documentation. Another benefit is that it would handle a rogue browser with a nonwhite default background color. With such a browser, if there were no explicit CSS rule for the white background color, then the user would see red, nonwhite, and blue. That isn't very patriotic for an American folk hero's quotes.

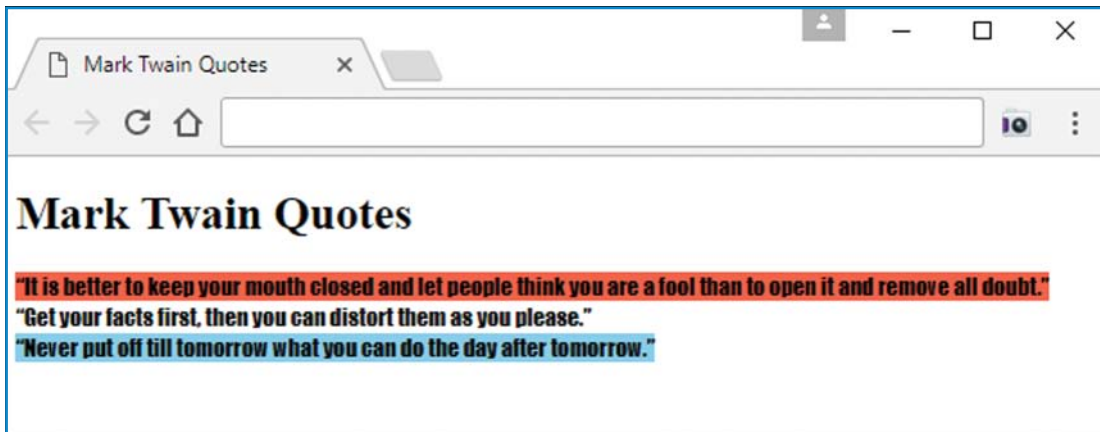


FIGURE 3.4 Mark Twain Quotes web page

class Selectors with Element Type Prefixes

Let's now discuss a specialized type of class selector—a class selector with an element type prefix. Here's the syntax:

```
element-type.class-value {property1: value; property2: value;}
```

And here's an example CSS rule that uses a class selector with an element type prefix:

```
q.blue {background-color: skyblue;}
```

Because `q.blue` has `.blue` in it, `q.blue` matches elements that have a `class` attribute value of "blue". But it's more granular than a standard class selector in that it looks for `class="blue"` only in `q` elements.

FIGURE 3.5 shows a modified version of the `style` container for the Mark Twain Quotes web page. It uses four class selectors with element type prefixes. How will that code change the appearance of the web page, compared to what's shown in Figure 3.4? The original `style` container used the simple class selector rule `.blue {background-color: skyblue;}`. That caused all elements with `class="blue"` to use the CSS color named `skyblue`. But suppose

```
<style>
h1.blue {color: blue;}
q.red {background-color: tomato;}
q.white {background-color: white;}
q.blue {background-color: skyblue;}
q {font-family: Impact;}
</style>
```

FIGURE 3.5 Improved style container for Mark Twain Quotes web page

you want a different shade of blue for the “Mark Twain Quotes” header. You could use a distinct class attribute value for the header, like “header-blue,” but having such a specific class attribute value would be considered poor style because it would lead to code that is harder to maintain. Specifically, it would be hard to remember a rather obscure name like “header-blue.” So, what’s the better approach? As shown in Figure 3.5, it’s better to use separate `h1.blue` and `q.blue` class selectors with element type prefixes. Note how the `h1.blue` rule specifies a background color of `blue`, and the `q.blue` rule specifies a background color of `skyblue`.

Figure 3.5’s `style` container uses a class selector with an element prefix, `q.red`, whereas the original `style` container used a simple class selector, `.red`. Because there’s only one element that uses `class="red"`, the `.red` class selector was sufficient by itself; however, using `q.red` (and also `q.white`) makes the code parallel for the three `q` element colors. More importantly, using a class selector with an element prefix makes the code more maintainable. *Maintainable* code is code that is relatively easy to make changes to in the future. For example, suppose you decide later that you want a different shade of red for an `h2` element. You can do that by using `q.red` and `h2.red`.

Class Selectors with * Prefixes

Instead of prefacing a class selector with an element type, as an alternative, you can preface a class selector with an `*`. Because `*` is the universal selector, it matches all elements. Therefore, the following CSS rule is equivalent to a standard class selector rule (with no prefix):

```
*.class-value {property1: value; property2: value;}
```

So what would the following CSS rule do?

```
*.big-warning {font-size: x-large; color: red;}
```

It would match all elements in the web page that have a class attribute value of `big-warning`, and it would display those elements with extra-large red font.

In the preceding CSS rule, note the hyphen in the `*.big-warning` class selector rule. The HTML5 standard does not allow spaces within `class` attribute values, so it would have been illegal to use `*.big warning`. If you want to use multiple words for a `class` attribute value, coding conventions suggest that you use hyphens to separate the words, as in `big-warning`.

CSS property names and CSS property values are built into the browser engine, so their naming is not subject to the discretion of web developers. Nonetheless, it’s still good to know their naming conventions so it’s easier to remember how to spell them. CSS property names follow the same coding convention as developer-defined `class` attribute values—if there are multiple words, use hyphens to separate the words (e.g., `font-size`). CSS property values usually follow the same use-hyphens-to-separate-multiple-words coding convention (e.g., `x-large` in the preceding code fragment). But sometimes nothing separates the words (e.g., `skyblue` in the Mark Twain Quotes web page).

3.7 ID Selectors

It's time for another type of selector—an ID selector. An ID selector is similar to a class selector in that it relies on an element attribute value in searching for element matches. As you might guess, an ID selector uses an element's `id` attribute (as opposed to a class selector, which uses an element's `class` attribute). A significant feature of an `id` attribute is that its value must be unique within a particular web page. That's different from a `class` attribute's value, which does not have to be unique within a particular web page. The ID selector's unique-value feature means that an ID selector's CSS rule matches only one element on a web page. This single-element matching mechanism is particularly helpful with links and with JavaScript, but we won't get to those things until later in the book. So why introduce the ID selector now instead of waiting for the links and JavaScript chapters? Because ID selectors are an important part of CSS.

Suppose you want the user to be able to link/jump to the “Lizard's Lounge” section of your web page. To do that, you'd need a link element (which we'll discuss in a later chapter) and also an element that serves as the target of the link. Here's a heading element that could serve as the target of the link:

```
<h3 id="lizards-lounge">Lizards Lounge</h3>
```

In this code, note the `id` attribute. The link element (not shown) would use the `id` attribute's value to indicate which element the user jumps to when the user clicks the link. For the jump to work, there must be no confusion as to which element to jump to. That means the target element must be unique. Using an `id` attribute ensures that the target element is unique.

Now that you have a rudimentary understanding of links and a motivation for using the `id` attribute, let's examine how to apply CSS formatting to an element with an `id` attribute. As always with CSS, you need a CSS rule. To match an element with an `id` attribute, you need an ID selector rule, and here's the syntax:

```
#id-value {property1: value; property2: value;}  
↑  
ID selector
```

The syntax is the same as for a class selector rule, except that you use a pound sign (`#`) instead of a dot (`.`), and you use an `id` attribute value instead of a `class` attribute value.

Remember the Lizard's Lounge heading element shown earlier? How would the following ID selector rule affect the appearance of the Lizard's Lounge heading?

```
#lizards-lounge {color: green;}
```

This rule would cause browsers to display the Lizards Lounge heading with green font.

Note the spelling of `lizards-lounge`. If you want to use multiple words for an `id` attribute value, the HTML5 standard states that it's illegal to use space characters to separate the words. Coding conventions suggest that you use hyphens to separate the words. That should sound familiar—`class` attribute values also use hyphens to separate words.

3.8 span and div Elements

So far, we’ve discussed different types of selectors—type selectors, the universal selector, class selectors, and ID selectors. No matter which selector you choose, you can apply it only if there’s an element in the web page body that matches it. But suppose you want to apply CSS to text that doesn’t coincide with any of the HTML5 elements. What should you do?

If you want to apply CSS to text that doesn’t coincide with any of the HTML5 elements, put the text in a `span` element or a `div` element. If you want the affected text embedded within surrounding text, use `span` (since `span` is a phrasing element). On the other hand, if you want the text to span the width of its enclosing container, use `div` (since `div` is a block element).

See **FIGURE 3.6** and note how the `div` and `span` elements surround text that doesn’t fit very well with other elements. Specifically, the `div` element surrounds several advertising phrases that describe Parkville’s world-famous Halloween on the River celebration, and the two `span` elements surround the two costs, \$10 and \$15. None of those things (a group of advertising phrases, a cost, and another cost) corresponds to any of the standard HTML elements, so `div` and `span` are the way to go if you want to apply CSS formatting.

The `div` and `span` elements are generic elements in that they don’t provide any special meaning when they’re used by themselves. They are simply placeholders to which CSS is applied. Think of `div` and `span` as vanilla ice cream and CSS as the various toppings you can add to the ice cream, such as chocolate chips, mint flavoring, and Oreos. Yummm!

In Figure 3.6, note the `span` element’s `class` attribute, copied here for your convenience:

```
<span class="white orange-background">$10</span>
```

In particular, note that there are two class selectors for the `class` attribute’s value—`white` and `orangebackground`. As you’d expect, that means that both the `white` and `orangebackground` CSS rules get applied to the `span` element’s content. Note that the two class selectors are separated with spaces. The delimiter spaces are required whenever you have multiple `class` selectors for one `class` attribute.

In the Pumpkin Patch web page, there are competing CSS rules for the two costs, \$10 and \$15. The `div` container surrounds the entire web page body, so it surrounds both costs, and it attempts to apply its orange text rule to both costs.¹ The first `span` container surrounds the first cost; consequently, the first `span` container attempts to apply its white text rule to the first cost. Likewise, the second `span` container surrounds the second cost; consequently, the second `span` container attempts to apply its black text rule to the second cost. So, what colors are used for the `span` text—white and black from the `span` containers or orange from the `div` container? As you can see in **FIGURE 3.7**’s browser window, the “\$10” cost text is white, and the



¹The “attempt to apply its orange text rule to both costs” is due to inheritance. We’ll introduce CSS inheritance formally in the next chapter.

“\$15” cost text is black. That means that the more local CSS rules (the two `span` rules) take precedence over the more global CSS rule (the `div` rule). The `span` rules are considered to be more *local* because their start and end tags immediately surround the cost content. In other words, their tags surround only their cost content and no other content. The `div` rule is considered to be more *global* because its start and end tags do not immediately surround the cost content. In other words,

```

<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<meta name="author" content="John Dean">
<title>Halloween on the River</title>
<style>
  .orange {color: darkorange;}
  .white {color: white;}
  .black {color: black;}
  .orange-background {background-color: orange;}
</style>
</head>
<body>
<div class="orange">
  Parkville's Halloween on the River, every weekend in October.<br>
  Corn maze: <span class="white orange-background">$10</span><br>
  All you can eat pumpkins:
  <span class="black orange-background">$15</span>
</div>
</body>
</html>

```

FIGURE 3.6 Source code for Pumpkin Patch web page

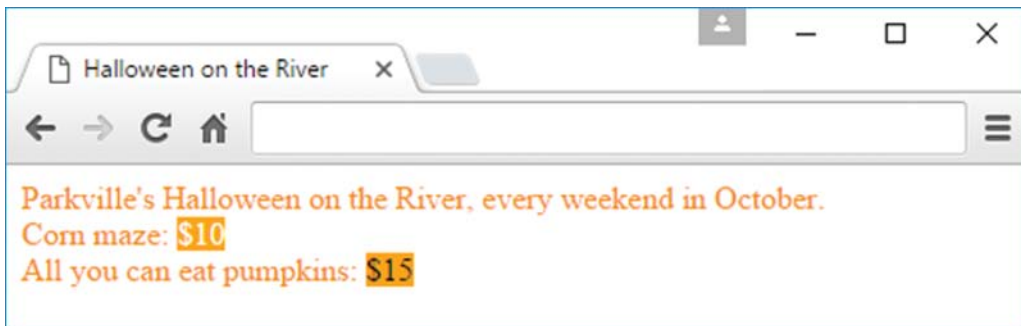


FIGURE 3.7 Pumpkin Patch web page

their tags surround not only the cost content, but also additional content. This *principle of locality*, where local things override global things, parallels the nature of the “cascading” that takes place in applying CSS rules. We’ll discuss that concept in the next section.

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

²A *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 <code>style</code> attribute.
2. In a <code>style</code> 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.