CHAPTER OUTLINE

3.1 Introduction

In the last chapter, we focused primarily on how to implement web page content. In this chapter, we focus on presentation of web page content. As you may recall, presentation refers to appearance and format. If you think appearance and format aren't all that important, think again. If your web page doesn't look good, people might go to it, but they'll leave quickly. An early exit might be OK if you're helping Grandma post her cat videos, but it's unacceptable for a business trying to generate revenue.

In this chapter, we start with an overview of Cascading Style Sheets (CSS) concepts and CSS basic syntax. We put those things into practice by applying CSS rules to various elements, including `span` and `div` elements. We show you how to position those rules (1) at the top of the web page's main file or (2) in an external file. In the second half of the chapter, we describe *CSS properties*. Properties are the hooks used to specify the appearance of the elements within a web page. Specifically, we introduce CSS properties for color, font, and line height. Also, we introduce CSS properties for borders, padding, and margins.

3.2 CSS Overview

The W3C's philosophy in terms of how HTML and CSS should fit together is (1) use HTML elements to specify a web page's content, and (2) use CSS to specify a web page's appearance. There are lots and lots of CSS properties that enable you to determine a web page's appearance. In this chapter, we'll cover quite a few of those properties, but not even close to all of them. When implementing a web page, if you need a particular format for an element and you can't find an appropriate CSS property in this book, don't give up right away. Search the Web for additional CSS properties to see if you can find one that suits your needs.

As you'll see shortly, and as you may recall from Figure 1.8's Kansas City Weather web page in Chapter 1, CSS code is normally separated from web page content code. Specifically, web page content code goes in the `body` container, whereas CSS code goes either at the top of the web page in the `head` container or in an external file. Why is that separation strategy a good thing? Because if you want to change the appearance of something, it's easy to find the CSS code—at the top of the web page or in an external file.

The current version of CSS is CSS3, and all major browsers support it. In 2009, the W3C started work on CSS4. There is no single, unified CSS4 specification. Instead, it's maintained as separate modules. Unfortunately, CSS4 is not fully supported by the major browsers yet. Thus, in this book, we stick with CSS3.

3.3 CSS Rules

The way CSS works is that CSS rules are applied to elements within a web page. Browsers determine which elements to apply the CSS rules to with the help of selectors. There are quite a few different types of selectors. For now, we'll introduce type selectors and the universal selector. Type selectors are very popular. The universal selector is not as popular, but it's important to understand it because you'll see it referred to on various websites, including the W3C's CSS website at https://www.w3.org/Style/CSS.

With a *type selector*, you use an element type (e.g., `hr`) to match all instances of that element type and then apply specified formatting features to those instances. For example, the following CSS rule uses a type selector with the `hr` element type and applies a width of 50% to all the `hr` elements in the current web page:

```
hr {width: 50%;}
```

A "width of 50%" means that for each `hr` element, its horizontal line will span 50% of the width of its enclosing container. Usually, but not always, the enclosing container will be the web page's `body` container.

Now for another type of selector—the universal selector. The *universal selector* uses the same syntax as the type selector, except that instead of specifying an element type, you specify `*`. The asterisk is a wildcard. In general, a *wildcard* is something that matches every item in a collection of things. For CSS selector rules, the `*` matches every element in a web page's collection of elements. Here's an example universal selector CSS rule that centers the text for every text-oriented element in the web page:

```
* {text-align: center;}
```

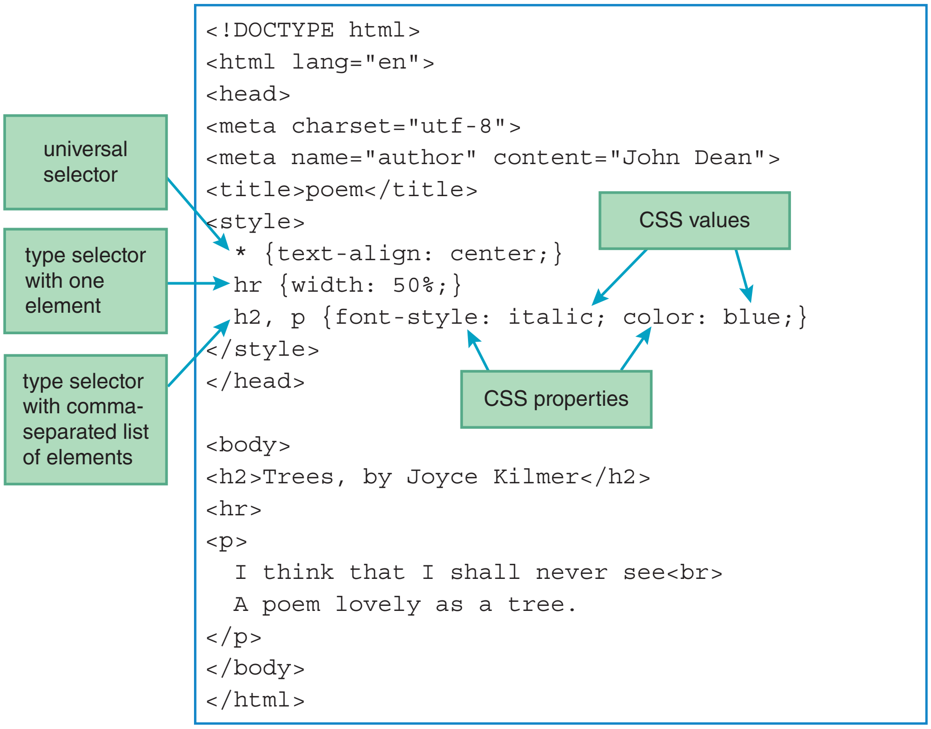
Even though the rule matches every element, because the property (`text-align`) deals with text, the rule affects only the elements that contain text.

3.4 Example with Type Selectors and the Universal Selector

Now let's look at a complete web page where we put into practice what's been covered so far in regard to CSS rules. Study the source code in FIGURE 3.1's Tree Poem web page. Notice the three CSS rules inside the `style` container. The first two rules should look familiar because they were presented in the previous section. The third rule uses a type selector with a slightly different syntax than before—there's a comma between two element types, `h2` and `p`. If you want to apply the same formatting feature(s) to more than one type of element, you can implement that with one rule, where the element types appear at the left, as part of a comma-separated list.

In Figure 3.1's three CSS rules, notice the four property-value pairs inside the { }'s, and copied here for your convenience:

- `text-align: center`
- `width: 50%`
- `font-style: italic`
- `color: blue`


```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<meta name="author" content="John Dean">
<title>poem</title>
<style>
  * {text-align: center;}
  hr {width: 50%;}
  h2, p {font-style: italic; color: blue;}
</style>
</head>

<body>
<h2>Trees, by Joyce Kilmer</h2>
<hr>
<p>
  I think that I shall never see<br>
  A poem lovely as a tree.
</p>
</body>
</html>
```
universal selector
type selector with one element
type selector with comma-separated list of elements
CSS values
CSS properties

FIGURE 3.1 **Source code for Tree Poem web page**

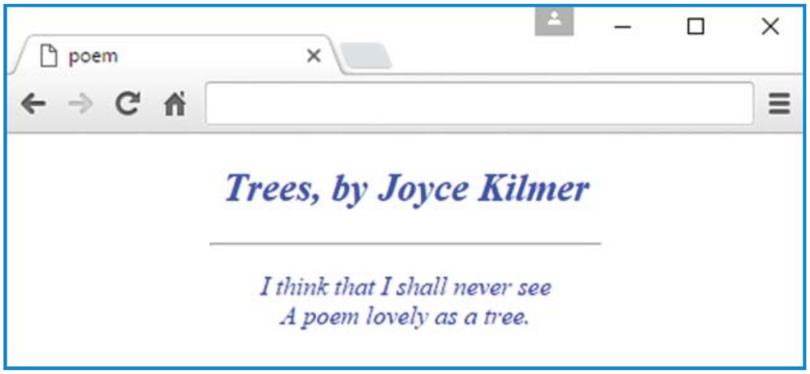


FIGURE 3.2 Tree Poem web page

We'll cover those properties in detail later on, but for now, go ahead and guess what they are for and how they affect the appearance of the Tree Poem web page. After you've made your guess, take a look at the resulting web page in **FIGURE 3.2**.

In the Tree Poem web page, the `* {text-align: center;}` rule causes the elements that contain text to be centered. The `hr` element does not contain text, so it's not affected by the `textalign` property. Nonetheless, as you can see, it's also centered. That's because `hr` elements are centered by default.

The `hr {width: 50%;}` rule causes the horizontal line to render with a width that's 50% of the web page `body`'s width.

Finally, the `h2, p {font-style: italic; color: blue;}` rule causes the heading and paragraph elements to be italicized and blue.

3.5 CSS Syntax and Style

CSS Syntax

In this section, we address CSS syntax details. First—the syntax for the `style` container. Refer back to Figure 3.1 and note how the three CSS rules are enclosed in a `style` container. Here's the relevant code:

```
<style>
  * {text-align: center;}
  hr {width: 50%;}
  h2, p {font-style: italic; color: blue;}
</style>
```

If you go back to the figure, you can see the `style` container positioned at the bottom of the web page's `head` container. It's legal to position it in the `body` container, but don't do it. Coding conventions suggest positioning it at the bottom of the web page's `head` container. By following that convention, other web developers will be able to find your CSS rules quickly.

In the `style` start tag, it's legal to include a `type` attribute with a value of `"text/css"`, like this:

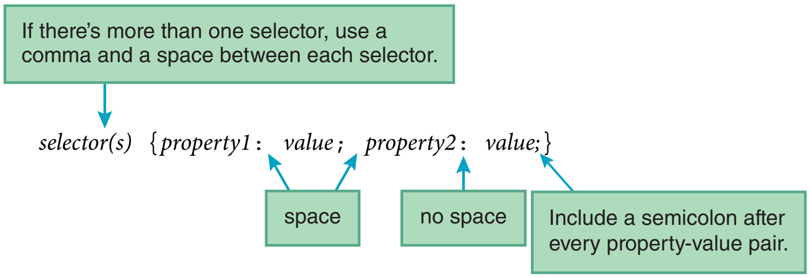
```
<style type="text/css">
```

In the Tree Poem web page, you can see that the `type` attribute is omitted. Currently, `text/css` is the only legal value for the `type` attribute, and it's the default value for the `type` attribute. So why did the HTML designers include a `type` attribute at all if there's only one type? They wanted to leave open the possibility of having different `style` types in the future. Google's Style Guide, which covers both HTML and CSS, recommends that you reduce the size of your web page file by omitting the `type` attribute, and we follow that convention in this book.

CSS Style

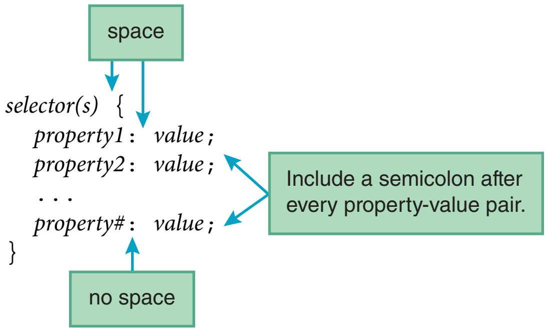
Now we'll look at some CSS guidelines that are not enforced by browsers or the HTML5 standard. They are style guidelines, and you should follow them so your code is easy to understand and maintain.

For short CSS rules, use this format:



If there's more than one selector, use a comma and a space between each selector.
selector(s) {property1: value; property2: value;}
space
no space
Include a semicolon after every property-value pair.

Remember in Chapter 2 when we introduced block formatting for multi-line container elements? That's where the start tag and end tag are aligned at the left, and interior lines are indented. Block formatting for CSS rules is similar in that the first and last lines are aligned at the left, and interior lines are indented. If you have a CSS rule that's kind of long (at least two or three property-value pairs), you should use block formatting like this:

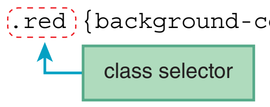


With both short and long CSS rules, the W3C CSS standard allows you to omit the semicolon after the last property-value pair. However, coding conventions suggest that you should not omit the last semicolon—you should include it. That way, if another property-value pair is added later on, there will be less likelihood of accidentally forgetting to add a semicolon in front of the new property-value pair.

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:



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:

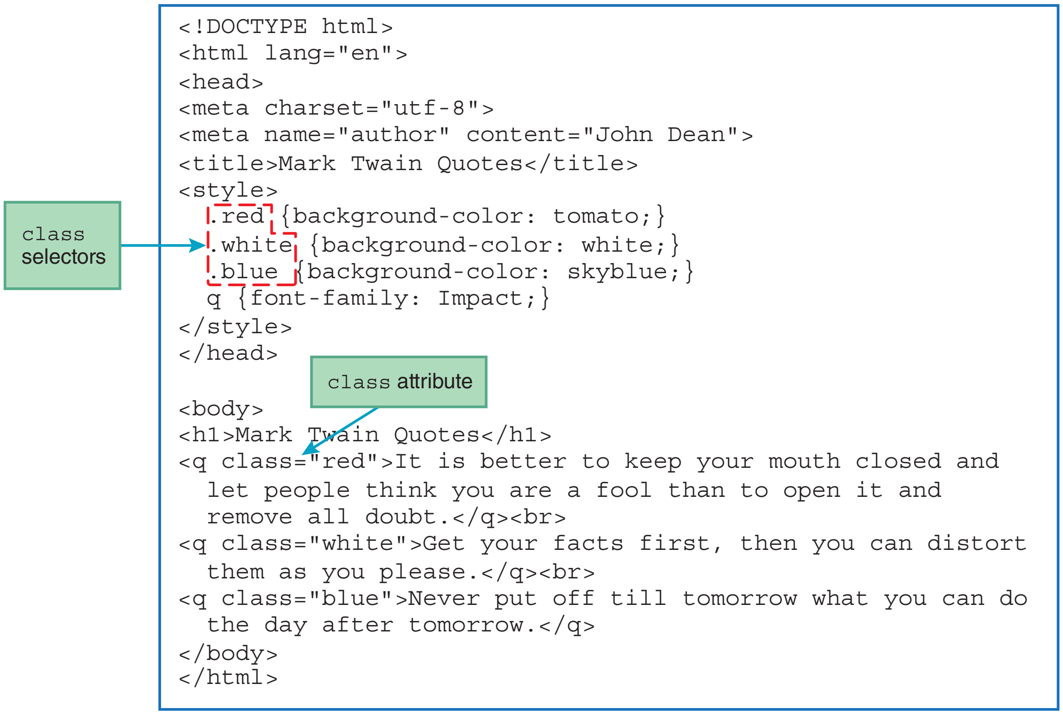
```
.red, .conspicuous, h1 {background-color: tomato;}
```

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 the 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>
```

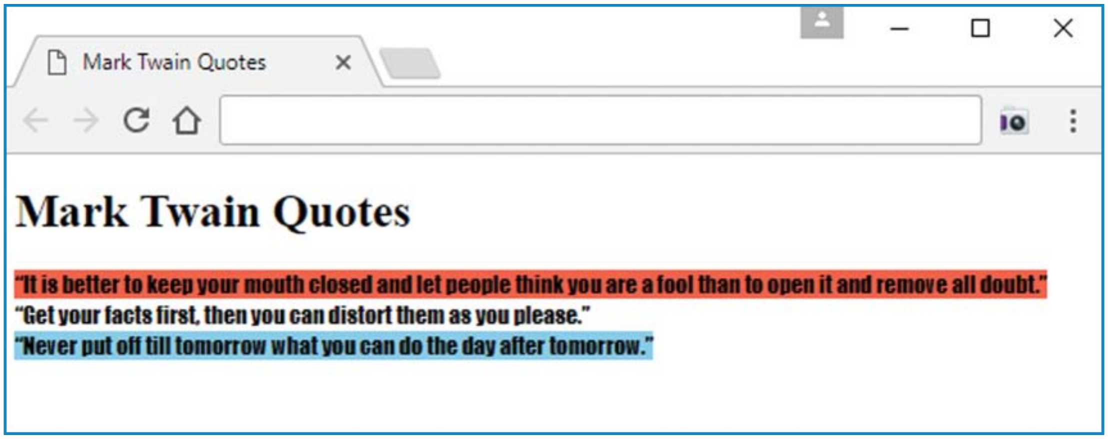
FIGURE 3.3 Source code for Mark Twain Quotes web page

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.



Mark Twain Quotes

"It is better to keep your mouth closed and let people think you are a fool than to open it and remove all doubt."
"Get your facts first, then you can distort them as you please."
"Never put off till tomorrow what you can do the day after tomorrow."

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

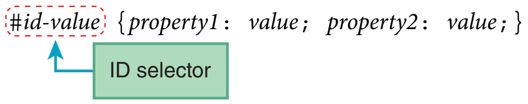
It’s time for another type of selector—an ID selector. An ID selector is similar to a class selector in that it relies on an element attribute value in searching for element matches. As you might guess, an ID selector uses an element’s `id` attribute (as opposed to a class selector, which uses an element’s `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*;}

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:

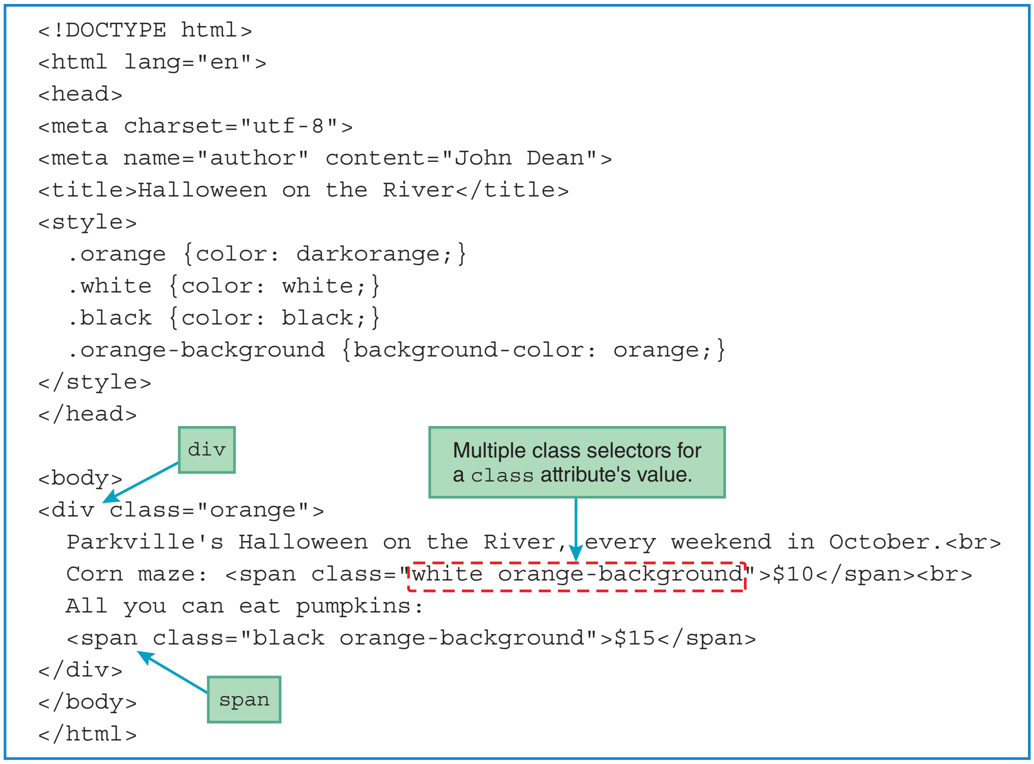
```
<span class="white aorange-background">$10</span>
```

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

[1] The "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,

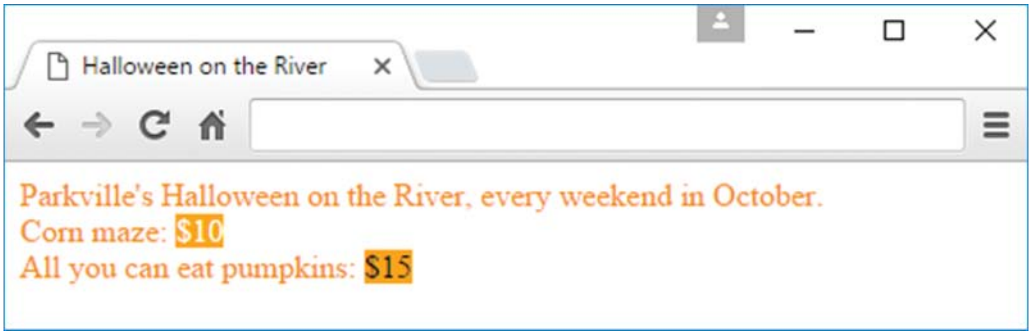


```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<meta name="author" content="John Dean">
<title>Halloween on the River</title>
<style>
  .orange {color: darkorange;}
  .white {color: white;}
  .black {color: black;}
  .orange-background {background-color: orange;}
</style>
</head>

<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>
```

div
Multiple class selectors for a class attribute's value.
span

FIGURE 3.6 **Source code for Pumpkin Patch web page**



Parkville's Halloween on the River, every weekend in October.
Corn maze: $10
All you can eat pumpkins: $15

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.