

Cascading Style Sheets

When you complete this chapter, you will be able to:

- ⦿ Recognize the benefits of using CSS
- ⦿ Build a basic style sheet
- ⦿ Use inheritance to write simpler style rules
- ⦿ Examine basic selection techniques
- ⦿ Apply basic selection techniques
- ⦿ Use class and id selectors
- ⦿ Use the `<div>` and `` elements
- ⦿ Use other selectors

Cascading Style Sheets (CSS) let you control the display characteristics of your Web site. In this chapter, you examine the syntax of CSS and learn how to combine CSS rules with your HTML code. You start by examining the CSS style rules, and then apply them to build a basic style sheet. You learn selection techniques to apply a particular style declaration to an element in your document. You learn about inheritance and how it affects your styles. You learn about two elements, `<div>` and ``, that were expressly created for use with CSS. More specific techniques include using the class attribute and id attribute to provide customized naming for your styles and applying the styles consistently across an entire Web site. You learn about using pseudo-selectors and pseudo-elements to select and apply CSS styles to links and to create special effects such as hovers, generated content, drop caps, and typographic effects.

Recognizing the Benefits of Using CSS

As you read in Chapter 1, Cascading Style Sheets offer many benefits to Web designers. CSS lets you separate style information from HTML, allowing you to provide style sheets for different destination media as your Web site requires. You can control the display characteristics of an entire Web site with a single style sheet, making maintenance and enhancements of display information a less taxing chore. You can express a wide range of style properties that increase the legibility, accessibility, and delivery of your content. You can build page layouts, either flexible or fixed, to suit your user and content needs. As you will see in this chapter and through the rest of the book, CSS is easy to learn and apply to your Web design projects.

The Evolution of CSS

Recall from Chapter 1 that Cascading Style Sheets were developed to standardize display information for Web pages and to enhance the separation of style and content. The first version of CSS, named CSS 1, was released in December 1996. CSS 1 contained properties to control fonts, color, text spacing and alignment, margins, padding, and borders. CSS 2, released in May 1998, added positioning, media types, and other enhancements. Even though CSS offered Web designers an easy-to-use and powerful method of controlling display properties, spotty browser support became a serious obstacle to the widespread adoption of the new style language. Finally, in May 2000, Internet Explorer (IE) 5 for the Macintosh became the first browser to fully support CSS 1.



Web sites such as www.webdevout.net/browser-support-css and westciv.com offer browser support cross-reference charts to help you determine which CSS properties are supported by different browser versions.

Later, as CSS became more fully supported, inconsistencies across different browsers still made Web development a matter of testing and retesting to make sure that CSS style rules were implemented correctly. IE 6 had serious problems interpreting CSS correctly, and even though few people still use IE 6, its legacy of inconsistent support affects CSS usage to this day. It is only with the release of Internet Explorer 8 that all major browsers support CSS 2.1, which was released in 2005.

The latest release of CSS is CSS level 3 (CSS3). Although work started on CSS3 in 1998, this version is still not a complete recommendation from the W3C in 2010. (In this context, a **recommendation** is the final stage of development by the World Wide Web Consortium (W3C), and means the release has been reviewed and tested extensively.) In CSS3, the W3C has broken CSS into modules, each of which contains a specific set of CSS properties. Some features of CSS3 are supported by the major browsers, but consistent support is not yet a reality. You can choose to implement CSS3 features, but make sure to test and retest for compatibility. You may decide to implement CSS3 features realizing that some browsers, especially IE 8, may not offer users the same result that they will see in other browsers such as Firefox, Safari, and Opera. In some cases these differences may be negligible, and the CSS3 feature can be implemented without seriously affecting the user's view of your content. CSS3 properties are always indicated as such throughout this book so you can make informed decisions about which properties to implement.

CSS Style Rules

In CSS, **style rules** express the style characteristics for an HTML element. A set of style rules is called a **style sheet**. Style rules are easy to write and interpret. For example, the following style rule sets all <p> elements in the document to blue text.

```
p {color: blue;}
```

A style rule is composed of two parts: a selector and a declaration. The style rule expresses the style information for an element. The **selector** determines the element to which the rule is applied. Selection is the key to working with CSS. As you will learn later in this chapter, CSS contains a variety of powerful selection techniques. By using the different types of selectors available, you build styles that affect an entire Web site or you can drill down to one specific paragraph or word in a Web page. The **declaration**, contained within curly brackets, details the exact property values. Figure 4-1 shows an example of a simple style rule that selects all <h1> headings and sets their color to red:

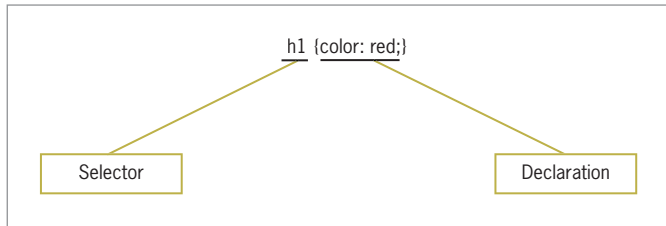


Figure 4-1 Style rule syntax

As illustrated in Figure 4-2, the declaration contains a property and a value. The **property** is a quality or characteristic, such as color, font size, or margin, followed by a colon (:). The **value** is the precise specification of the property, such as blue for color, 125% for font size, or 30 px (pixels) for margin, followed by a semicolon (;). CSS contains a wide variety of properties, each with a specific list of values. As you will see later in this chapter, you can combine selectors and property declarations in a variety of ways.

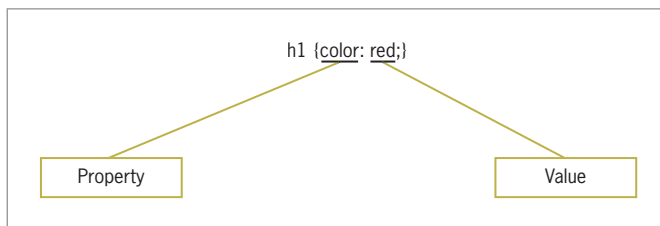


Figure 4-2 Property declaration syntax

This chapter uses a variety of CSS style rules as examples. Although you have not yet learned about their properties in detail, you will see that the CSS property names express common desktop publishing characteristics such as font-family, margin, text-indent, and so on. The property values sometimes use abbreviations such as *px* for pixel, percentages such as 200%, or keywords such as *bold*. You will learn about these properties and values in detail as you progress through this book.

Combining CSS Style Rules with HTML

You can combine CSS rules with HTML code in the following three ways:

- Inline style
- Internal style sheet
- External style sheet

Figure 4-3 shows the code for a single Web page with all three methods of combining CSS style rules with the HTML.

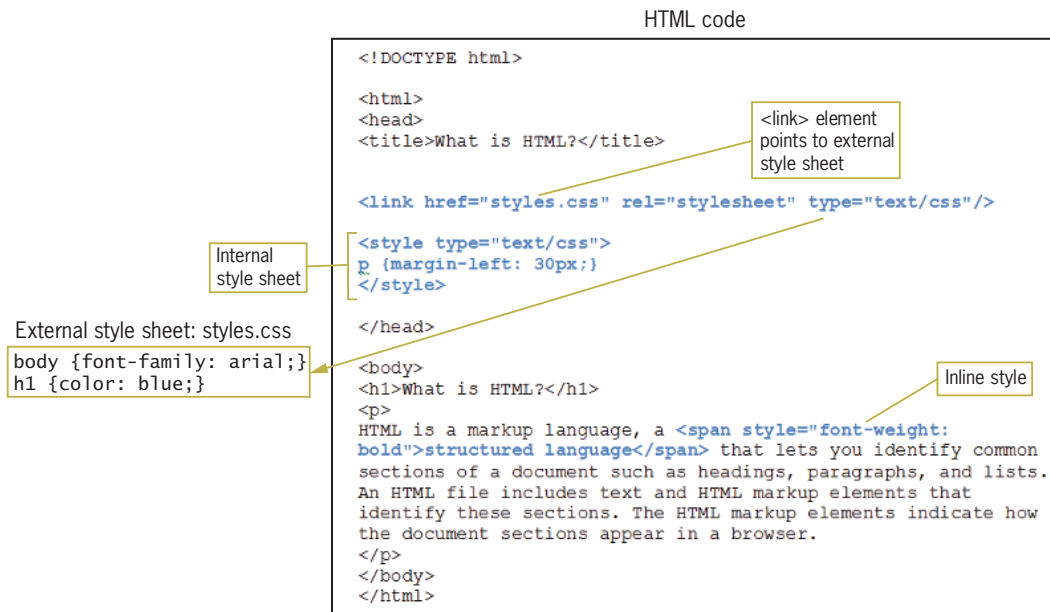


Figure 4-3 Three methods of combining CSS with HTML

As you review Figure 4-3, keep the following principles in mind:

- The `<link>` element in the document's `<head>` section points to an external style sheet named `styles.css`, which contains rules that set the body text to Arial and the `<h1>` heading to the color blue. This external style sheet can be linked from other documents as well, meaning these styles will be applied across multiple pages within the Web site.
- The internal style sheet affects only this one document in which it is contained, making all paragraph elements within the document have a 30-pixel left margin.
- The inline style within the paragraph affects only the span of words it surrounds, making the words *structured language* bold.

In this example, the styles rules from all three sources—external, internal, and inline—combine to create the finished look for this Web page. This flexibility in the application of style rules is an enormous benefit to Web designers, who can set styles that apply across an entire Web site using external style sheets, while maintaining control over individual pages and elements if necessary. Each method is discussed in detail in the following sections.

Using External Style Sheets

Placing style sheets in an external document lets you specify rules for multiple Web pages. This is an easy and powerful way to use style sheets because it lets you control the styles for an entire Web site with one style sheet file. Additionally, external style sheets are stored in the user's cache, so once downloaded, they are referenced locally for every file on your Web site, saving time for your user.

An external style sheet is simply a text document that contains the style rules. External style sheets have a .css extension. Here's an example of a simple external style sheet named styles.css:

```
h1 {color: white; background-color: green;}  
h2 {color: red;}
```

The style sheet file does not contain any HTML code, just CSS style rules, because the style sheet is not an HTML document. It is not necessary to use the <style> element in an external style sheet.

Linking to an External Style Sheet

The **<link> element** lets you establish document relationships. It can only be used within the <head> section of a document. To link to an external style sheet, add the <link> element, as shown in the following code:

```
<head>  
<title>Sample Document</title>  
<link href="styles.css" rel="stylesheet" type="text/css" />  
</head>
```

The <link> element in this code tells the browser to find the specified style sheet. The href attribute states the relative URL of the style sheet. The rel attribute specifies the relationship between the linked and current documents. The browser displays the Web page based on the CSS display information.

The advantage of the external style sheet is that you can state the style rules in one document and affect all the pages on a Web site. When you want to update a style, you only have to change the style rule once in the external style sheet.

Using Internal Style Sheets

Use the `<style>` element to create an internal style sheet in the `<head>` section of the document. Style rules contained in an internal style sheet only affect the document in which they reside. The following code shows a `<style>` element that contains a single style rule:

```
<head>
<title>Sample Document</title>
<style type="text/css">
h1 {color: red;}
</style>
</head>
```

In this code sample, note the type attribute to the `<style>` element. The value “text/css” defines the style language as Cascading Style Sheets.

Using Inline Styles

You can define the style for a single element using the style attribute, which is called an inline style.

```
<h1 style="color: blue">Some Text</h1>
```

You generally use the style attribute to override a style that was set at a higher level in the document, such as when you want a particular heading to be a different color from the rest of the headings on the page. (You’ll learn more about overriding a style later in this chapter.) The style attribute is also useful for testing styles during development. You will probably use this method of styling an element the least, because it only affects one instance of an element in a document.

Writing Clean CSS Code

When you are creating external or internal style sheets, it is best to write CSS code that is consistent and easy to read. The flexibility of CSS syntax lets you write style rules in a variety of ways. For example, you’ve already seen style rules in this chapter that combine multiple declarations on the same line, as in the following code:

```
h1 {color: white; background-color: green;}
```

This technique is fine if you have simple styles, but style rules often contain many declarations that would be too hard to read in this format, especially when styles grow more complex, as shown in the following code:

```
p {font-family: arial, helvetica, sans-serif; font-size: 85%;
line-height: 110%; margin-left: 30px;}
```

This style rule is easier to read and maintain if you use a cleaner, more organized format as shown in the following code:

```
p {
  font-family: arial, helvetica, sans-serif;
  font-size: 85%;
  line-height: 110%;
  margin-left: 30px;
}
```

This format makes it easier to read the style rules, because all properties are consistently left-aligned with their values on the right.

Using Comments

CSS allows comments within the `<style>` element or in an external style sheet. CSS comments begin with the forward slash and asterisk characters (`/*`) and end with the asterisk and forward slash characters (`*/`). You can use comments in a variety of ways, as shown in the following code:

```
<style type="text/css">
/* This is the basic style sheet */
h1 {color: gray;} /* The headline color */
h2 {color: red;} /* The subhead color */
</style>
```

Comments provide documentation for your style rules. Because they are embedded directly in the style sheet, they provide immediate information to anyone who needs to understand how the style rules work. Comments are always useful, and you should consider using them in all of your code, whether as a simple reminder to yourself or as an aid to others with whom you work.

Activity: Building a Basic Style Sheet

In the following steps, you build and test a basic style sheet. Save your file and test your work in your browser as you complete each step. The new code to add is displayed in blue text. Refer to Figure 4-4 as you progress through the steps to see the results.

To build a basic style sheet:

1. Copy the **basic.htm** file from the Chapter04 folder provided with your Data Files to the Chapter04 folder in your work folder. (Create the Chapter04 folder, if necessary.)

2. In your browser, open **basic.htm**. When you open the Web page, it looks like Figure 4-4.

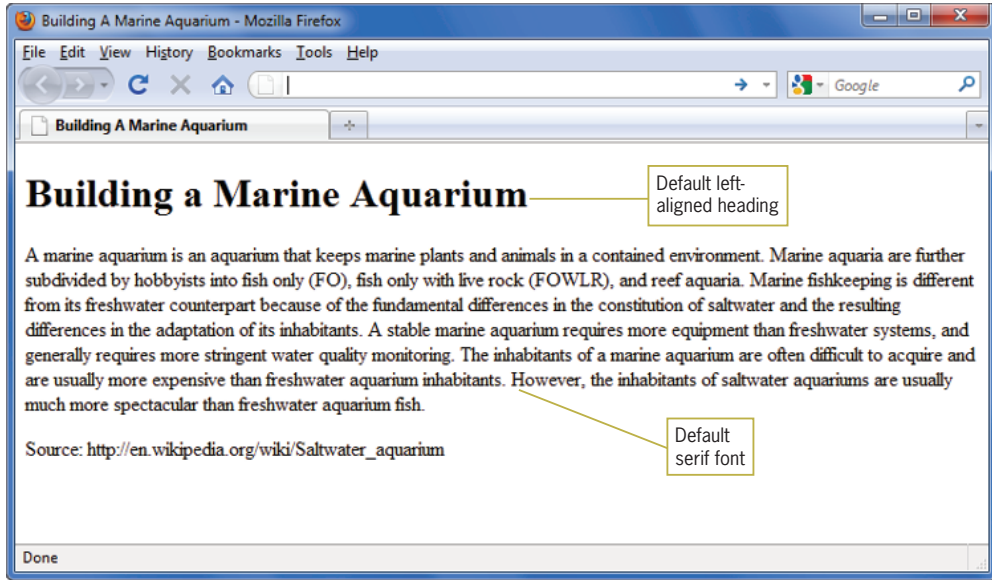


Figure 4-4 Original Web page with default styling

3. Open the **basic.htm** file in your HTML editor and examine the code. Notice that the file contains basic HTML code with no style information.
4. Add a `<style>` element in the `<head>` section to contain your style rules, as shown in blue in the following code. Leave a line or two of white space between the `<style>` tags to contain the style rules.

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

</style>
</head>
```

5. Add a style rule for the `<h1>` element, as shown in blue in the following code. This style rule uses the `text-align` property to center the heading.

```
<head>
<style type="text/css">
h1 {text-align: center;}
</style>
</head>
```

6. Save the file as **basic.htm** in the Chapter04 folder in your work folder, and then reload the file in the browser. The `<h1>` element is now centered.

7. Add a style rule for the `<p>` element, shown in blue in the following code. This style rule uses the `font-family` property to specify a sans-serif font for the paragraph text. You will learn more about the `font-family` property in Chapter 5.

```
<head>
<style type="text/css">
h1 {text-align: center;}
p {font-family: sans-serif;}
</style>
</head>
```

8. Save the file and then reload it in the browser. Figure 4-5 shows the finished Web page. Notice that the `<p>` element is now displayed in a sans-serif typeface and the `<h1>` heading is centered.

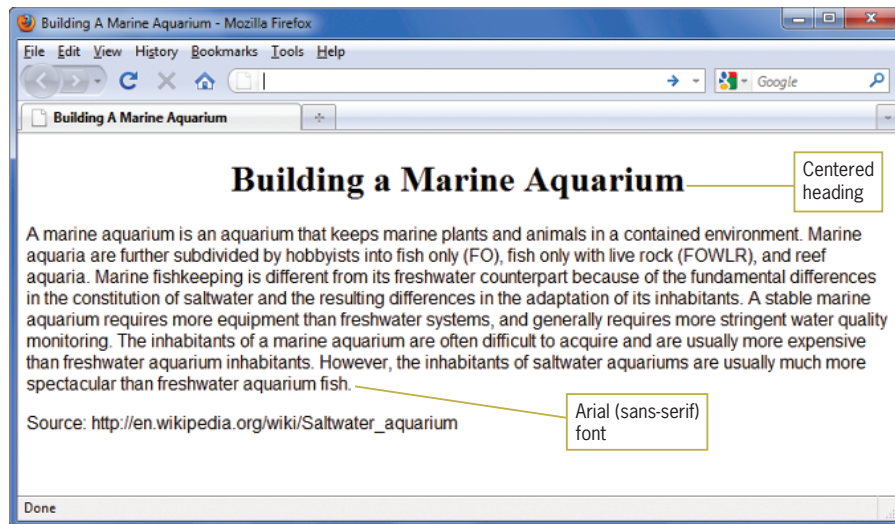


Figure 4-5 Finished Web page styled with CSS

Using Inheritance to Write Simpler Style Rules

The elements in an HTML document are structured in a hierarchy of parent and child elements. Figure 4-6 represents the structure of a simple HTML document.

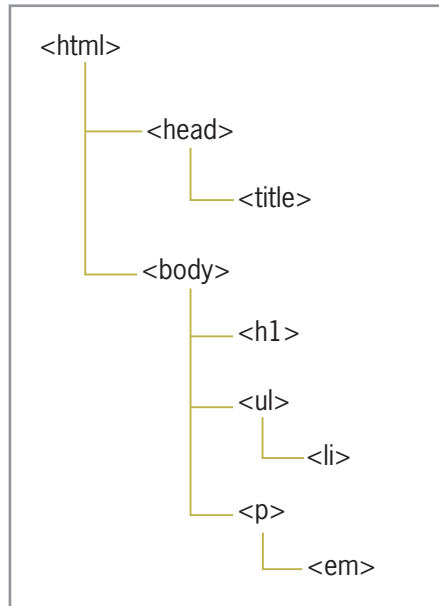


Figure 4-6 HTML document structure

Note the hierarchical structure of the elements. At the top, `<html>` is the parent element of the document. **Parent elements** contain nested elements called **child elements**. Both `<head>` and `<body>` are immediate child elements of `<html>`. Yet, `<head>` and `<body>` are parent elements as well, because they contain other nested elements. As you travel further down the document hierarchy, you find additional elements that are both parent and child elements, such as `<p>` and ``.

By default, most CSS properties inherit from parent elements to child elements, which is called **inheritance**. The CSS property descriptions in the following chapters and Appendix B list whether a property is inherited. Therefore, if you set a style rule for `` elements in the document shown in Figure 4-6, the `` elements inherit the style rules for ``, unless you specifically set a rule for ``.

You can style multiple document elements with just a few style rules if you let inheritance work for you. For example, consider the following set of style rules for a document.

```
<style type="text/css">
h1 {color: red;}
p  {color: red;}
ul {color: red;}
```

```
em {color: red;}  
li {color: red;}  
</style>
```

This style sheet sets the color to red for five different elements in the document. Inheritance lets you write a far simpler rule to accomplish the same results:

```
<style type="text/css">  
body {color: red;}  
</style>
```

This rule works because all of the elements are children of `<body>` and because all the rules are the same. It is much more efficient to write a single rule for the parent element and let the child elements inherit the style. Because `<body>` is the parent element of the content area of the HTML file, it is the selector to use whenever you want to apply a style across the entire document.

Examining Basic Selection Techniques

In this section you will review style rule syntax and learn about the following basic selection techniques:

- Using type selectors
- Grouping selectors
- Combining declarations
- Using descendant selectors

Using Type Selectors

As you learned previously, the selector determines the element to which a style declaration is applied. To review, examine the syntax of the style rule shown in Figure 4-7. This rule selects the `<h1>` element in the document and sets the text color to red.

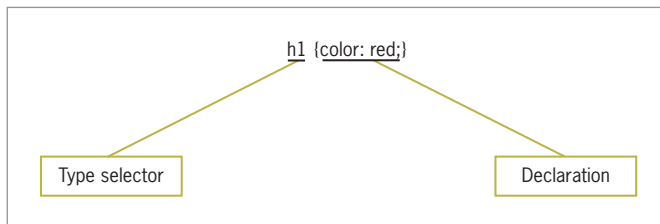


Figure 4-7 Style rule syntax

This rule uses a **type selector** to apply the rule to every instance of the element in the document. This is the simplest kind of selector, and many style sheets are composed primarily of type selector style rules, as shown in the following code:

```
body {color: gray;}
h2 {color: red;}
p {font-size: 85%;}
```

Grouping Selectors

To make your style rules more concise, you can group selectors to which the same rules apply. For example, the following style rules set the same declaration for two different elements—they set the color of <h1> and <h2> elements to red:

```
h1 {color: red;}
h2 {color: red;}
```

These two style rules can be expressed in a simpler way by separating the selectors with commas:

```
h1, h2 {color: red;}
```

Combining Declarations

In many instances you want to state multiple property declarations for the same selector. The following style rules set the <p> element to 12-point, blue text:

```
p {color: blue;}
p {font-size: 125%;}
```

These two style rules can be expressed in a simpler fashion by combining the declarations in one rule. The declarations are separated by semicolons:

```
p {
  color: blue;
  font-size: 125%;
}
```

Using Descendant Selectors

A descendant selector (sometimes known as a contextual selector) is based on the hierarchical structure of the elements in the document tree. This selector lets you select elements that are the descendants of other elements. For example, the following rule selects only elements that are contained within <p> elements. All other elements in the document are not affected.

```
p em {color: blue;}
```

Notice that the selector contains multiple elements, separated only by white space. You can use more than two elements if you prefer to choose more specific selection characteristics. For example, the following rule selects `` elements within `` elements within `` elements only:

```
ul li em {color: blue;}
```

Using the Universal Selector

The **universal selector** lets you quickly select groups of elements and apply a style rule. The symbol for the universal selector is the asterisk (`*`). For example, to set a default color for all elements within a document, use the following rule:

```
* {color: purple;}
```

You can also use the universal selector to select all children of an element. For example, the following rule sets all elements within a `<div>` element to a sans-serif typeface:

```
div * {font-family: sans-serif;}
```

Remember that the universal selector is always overridden by more specific selectors. The following style rules show a universal selector along with two other rules that have more specific selectors. In this example, the `<h1>` and `<h2>` rules override the universal selector for the `<h1>` and `<h2>` elements.

```
* {color: purple;}
h1 {color: red;}
h2 {color: black;}
```

Activity: Applying Basic Selection Techniques

In the following steps, you will build a style sheet that uses basic selection techniques. Save your file and test your work in your browser as you complete each step. The new code to add is displayed in blue text. Refer to Figure 4-8 as you progress through the steps to see the results.

To build the style sheet:

1. Copy the **oz.htm** file from the Chapter04 folder provided with your Data Files to the Chapter04 folder in your work folder.
2. Open the file **oz.htm** in your HTML editor, and save it as **oz1.htm** in the same location.
3. In your browser, open the file **oz1.htm**. When you open the file, it looks like Figure 4-8.

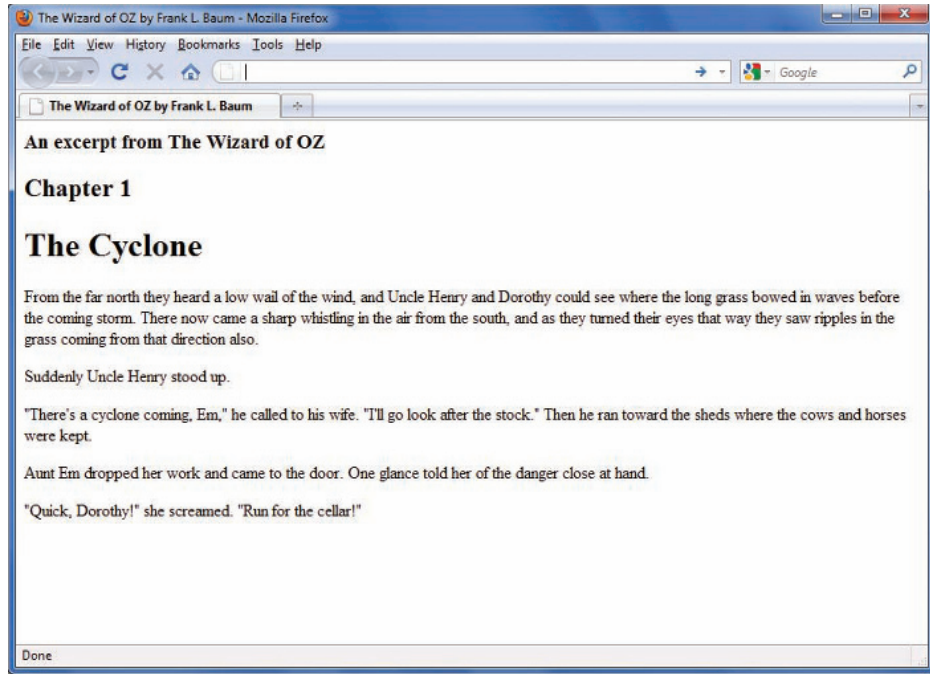


Figure 4-8 Original oz1.htm Web page

4. In your text editor, examine the page code. Notice that the file contains basic HTML code with no style information.
5. Add a `<style>` element in the `<head>` section to contain your style rules, as shown in blue in the following code. Leave a line or two of white space between the `<style>` tags to contain the style rules.

```
<head>
<title>The Wizard of OZ by Frank L. Baum</title>
<style type="text/css">

</style>
</head>
```

6. Write the style rule for the `<h3>` element. The requirements for this element are right-aligned text. The style rule looks like this:

```
<style type="text/css">
h3 {
    text-align: right;
}
</style>
```

7. Write the style rules for the `<h1>` and `<h2>` elements, which share some common property values. Both elements

have a left margin of 20 pixels (abbreviated as 20px) and a sans-serif font style. Because they share these properties, group the two elements to share the same style rule, as shown in the following code:

```
<style type="text/css">
h3 {
    text-align: right;
}

h1, h2 {
    margin-left: 20px;
    font-family: sans-serif;
}

</style>
```

8. Write an additional rule for the <h1> element. The <h1> element has two style properties that it does not share with <h2>, so a separate style rule is necessary to express the border and padding white space within the border. This rule uses the border shortcut property to specify multiple border characteristics—a 1-pixel border weight and solid border style. The padding-bottom property sets the border 5 pixels below the text.

```
<style type="text/css">
h3 {
    text-align: right;
}

h1, h2 {
    margin-left: 20px;
    font-family: sans-serif;
}

h1 {
    border-bottom: 1px solid;
    padding-bottom: 5px;
}

</style>
```

9. Write a style rule for the <p> elements so they have a 20-pixel left margin (to line up with the other elements on the page), Arial font style, and a font size of 120%.

```
<style type="text/css">
h3 {
    text-align: right;
}

h1, h2 {
    margin-left: 20px;
    font-family: sans-serif;
}
```



```

h1 {
    border-bottom: 1px solid;
    padding-bottom: 5px;
}

p {
    margin-left: 20px;
    font-family: arial;
    font-size: 120%;
}

</style>

```

Figure 4-9 shows the finished document with the style properties.

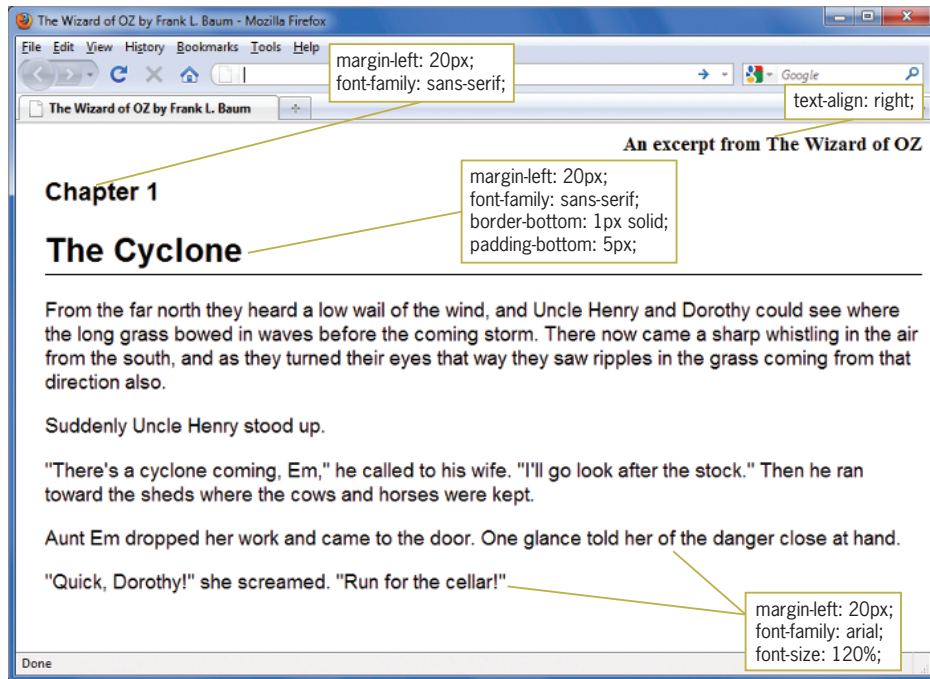


Figure 4-9 Finished Web page styled with CSS rules

Using Class and ID Selectors

This section describes CSS selection techniques that allow more than the basic element-based selection capabilities described in the previous section. You will learn to select elements of an HTML document using the following methods:

- The class selector
- The id selector

- The <div> and elements
- The pseudo-class and pseudo-element selectors

Using the Class Selector

The class selector lets you write rules, give them a name, and then apply that name to any elements you choose. You apply the name using the class attribute, which is a common attribute that applies to any HTML element. Refer to Appendix A for descriptions of the common attributes. To apply a style rule to an element, you can add the class attribute to the element and set it to the name you have specified.

To create a class, declare a style rule. The period (.) flag character indicates that the selector is a class selector. Figure 4-10 shows an example of a rule with a class selector named *special*.

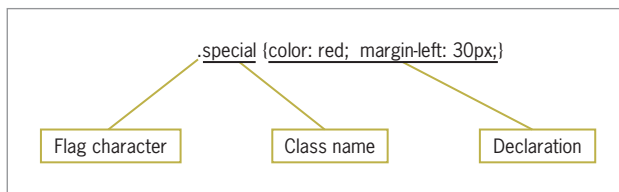


Figure 4-10 Class syntax

After writing the style rule, add it to the document by using the class attribute, as shown in the following code:

```
<h1 class="special">This heading will be red with a
30-pixel left margin.</h1>
```

The class attribute lets you select elements with greater precision. For example, read the following style rule:

```
p {font-size: 85%;}
```

This rule sets all <p> elements in the document to a font size of 85%. Now suppose that you want one <p> element in your document to have a different style characteristic, such as bold text. You need a way to specifically select that one paragraph. To do this, use a class selector. The following style rule sets the style for the class named *intro*:

```
.intro {font-size: 125%; font-family: sans-serif;}
```

The class selector can be any name you choose. In this instance, the class name *intro* denotes a special paragraph of the document. Now apply the rule to the `<p>` element in the document using the class attribute:

```
<p class="intro">This is the first paragraph of the
document. It has a different style based on the "intro"
class selector.</p>
```

```
<p>This is the second paragraph of text in the document.
It is a standard paragraph without a class attribute.</p>
```

Figure 4-11 shows the result of the style rule.

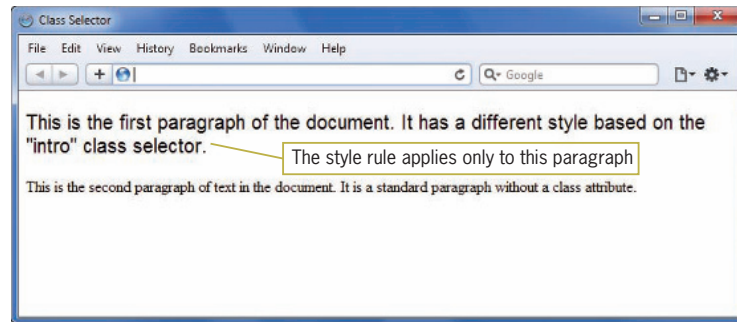


Figure 4-11 Styling with a class attribute

Making Class Selectors More Specific

Using the class attribute is a powerful selection technique, because it allows you to write style rules with names that are meaningful to your organization or information type. The more specific your class names become, the greater control you need over the way they are applied. In the preceding example, you saw a style rule named *intro* that was applied to a `<p>` element. However, the *intro* style can be applied to any element in the document, not just `<p>`. To solve this problem, you can restrict the use of the class attribute to a single element type.

For example, your organization might use a special style for a procedure heading, the heading that appears before steps in a training document. The style is based on an `<h1>` element, with a sans-serif font and left margin of 20 pixels. Everyone in your organization knows this style is named *procedure*. You can use this same style name in your style sheet, as shown in the following style rule:

```
.procedure {
    font-family: sans-serif;
    margin-left: 20px;
}
```

To use these rules in the document, you apply the class attribute, as shown in the following code:

```
<h1 class="procedure">Procedure Heading</h1>
```

This works well, but what happens if someone on your staff neglects to apply the classes properly? For the style rule to work, it must be applied to an `<h1>` element. To restrict the use of the class to `<h1>` elements, include a prefix for the class selector with the element to which you want it applied:

```
h1.procedure {
    font-family: sans-serif;
    margin-left: 20px;
}
```

These style rules restrict the use of the *procedure* style to `<h1>` elements only. If this style is applied to other elements it will not work.

Using the id Selector

The id attribute, like the class attribute, is an HTML common attribute. The difference between id and class is that it should refer to only one occurrence within a document. The id attribute has become the selector of choice when identifying layout sections of the page. The id attribute is perfect for this because generally only one layout section such as a footer or sidebar will occur per page. For example, you might want to specify that only one `<p>` element can have the id *copyright* and its associated style rule. Figure 4-12 shows a style rule that uses the id *copyright* as a selector.

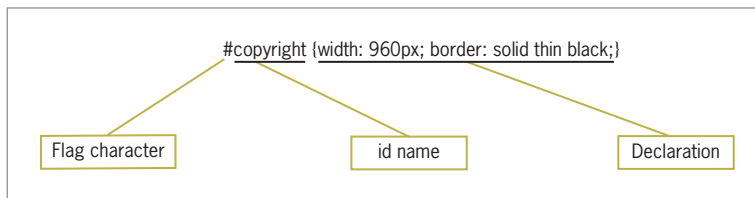


Figure 4-12 Using the id selector

Notice that the id selector uses a pound sign (`#`) flag character instead of the period you used with the class selector. You can apply the id value to the appropriate element in the document, in this example a `<p>` element:

```
<p id="copyright">This is the copyright information for  
the page.</p>
```

The id value uniquely identifies this one `<p>` element as containing copyright information. For consistency in design, no other element in the document should share this exact id value.

Just like classes, you can make id selectors more specific by adding an element selector before the flag character, as shown in the following code:

```
p#copyright {  
    font-family: times;  
    text-align: center;  
}
```

This style rule will only apply to a `<p>` element with the id set to *copyright*.

Using the `<div>` and `` Elements

The `<div>` (division) and `` (span of words) elements are designed to be used with CSS. They let you specify logical divisions within a document that have their own name and style properties. The difference between `<div>` and `` is their element display type, which is described in more detail in Chapter 6. `<div>` is a block-level element, and `` is its inline equivalent. Used with the class and id attributes, `<div>` and `` let you effectively create your own element names for your HTML documents.

Working with `<div>` Elements

You can use the `<div>` element with the class and id attributes to create logical divisions on a Web page, such as headers, footers, columns, and so on. The `<div>` element can contain other block level elements such as paragraphs or even other divisions. In modern Web design, divisions are the main content containers on a Web page, replacing the use of tables.

To create a customized division, declare it with a class or id selector in the style rule. The following example specifies a division with an id named *column* as the selector for the rule:

```
div#column {  
    width: 200px;  
    height: auto;  
    padding: 15px;  
    border: thin solid;  
}
```

To apply this rule, specify the <div> element in the document. Then use the id attribute to specify the exact type of division. In the following example, the code defines the <div> element with the id named *column*:

```
<div id="column">This division displays as a column of
text in the browser window. This is one of the uses of the
division element as a page layout tool with CSS. You will
learn more about this in later chapters of this book. </div>
```

Figure 4-13 shows the result of the style rule.

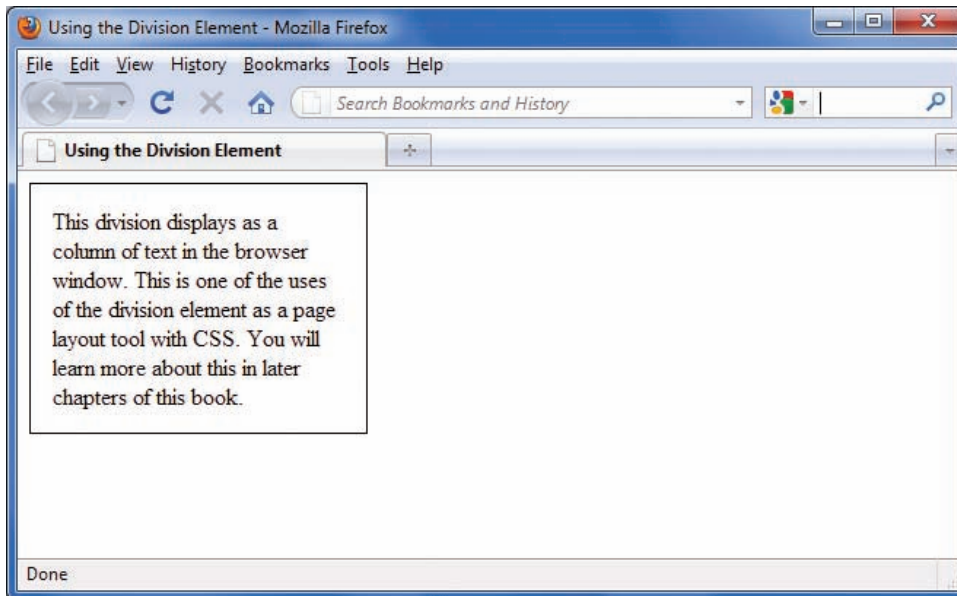


Figure 4-13 Using the <div> element to create a column of text

Working with Elements

The element lets you specify inline elements within a document that have their own name and style properties. Inline elements reside within a line of text, like the or element. You can use with a class or id attribute to create customized inline elements.

To create a span, declare it within the <style> element first. The following example specifies a span named *logo* as the selector for the rule:

```
span.logo {
    color: white;
    background-color: black;
}
```

Next, specify the `` element in the document. Then use the class attribute to specify the exact type of span. In the following example, the code defines the `` element as the class named *logo*.

```
<p>Welcome to the <span class="logo">Wonder Software</span>
Web site.</p>
```

Figure 4-14 shows the result of the style rule.



Figure 4-14 Using the `` element

Using Other Selectors

Besides class and id selectors, you can use attribute selectors, pseudo-class and pseudo-element selectors, and CSS3 selectors.

Using Attribute Selectors

Attribute selectors let you select an element based on whether the element contains an attribute. You can also choose an element based on a specific value the attribute contains.

Attribute selectors match against attributes and their values. In the following code, the element has three attributes: `src`, `title`, and `alt`.

```

```

Using attribute selectors, you could choose this based on the presence of the `title` attribute, as in the following code:

```
img[title] {border-color: red;}
```

Or, you could choose this based on the value that the `title` attribute contains, as shown:

```
img[title=home] {border-color: red;}
```

Attribute selectors come in handy when there are multiple elements that share the same characteristics, sometimes only differing by the values of the attributes they contain. Table 4-1 lists the CSS 2.1 attribute selectors, which would be more commonly supported by most browsers. CSS3 selectors are described later in this chapter.

Syntax	Description	Example
[attribute]	Select when the element contains the named attribute; the attribute value is not matched	p[title] {color: blue;} matches: <p title="opening"> <p title="closing">
[attribute=value]	Select when the element contains the named attribute and specific value	p[title=footer] matches: <p title="footer">
[att~=val]	Select when the attribute contains the value in a list of white-space separated values	p[att~=copyright] matches: <p type="copyright trademark">
[att =val]	Select when an attribute contains the exact value or begins with the value	p[att en] matches: <p lang="english">

Table 4-1 CSS 2.1 Attribute Selectors

Using Pseudo-Class and Pseudo-Element Selectors

Pseudo-class and pseudo-element selectors let you express style declarations for characteristics of a document that are not signified with the standard HTML elements. **Pseudo-classes** select elements based on characteristics other than their element name. For example, assume that you want to change the color of a new or visited hypertext link. No HTML element directly lets you express these characteristics of the <a> element. With CSS, you can use the pseudo-class selector to change the link color.

Pseudo-elements let you change other aspects of a document that are not classified by elements, such as applying style rules to the first letter or first line of a paragraph. For example, you might want to create a drop initial or drop capital that extends below the line of type, or make the first line of a paragraph all uppercase text. These are common publishing design techniques that are not possible with standard HTML code. With CSS you can use the :first-letter and :first-line pseudo-elements to add these two style characteristics to your documents.

Using the Link Pseudo-Classes

The link pseudo-classes let you change the style characteristics for four different hypertext link states, as described in Table 4-2.

Pseudo-Class	Description
:link	Selects any unvisited link that the user has not clicked or is not hovering over with their mouse pointer
:visited	Selects any link that your user has already visited
:hover	Selects any link that your user is pointing to with the mouse pointer
:active	Selects a link for the brief moment that your user is actually clicking the link

Table 4-2 Link Pseudo-Classes

The following rules change the colors of the hypertext links in a Web page:

```
a:link {color: red;}
a:visited {color: green;}
```

Note that the colon (:) is the flag character for a pseudo-class.



Because of the specificity of the pseudo-class selectors, you should always place your link pseudo-class in the following order:

1. Link
2. Visited
3. Hover
4. Active

This order is sometimes abbreviated to LVHA. You do not have to use all of the different link states, so if you skip one make sure the others follow the correct order. You will learn about specificity later in this chapter.

Whether you choose to change your hypertext link colors depends on the design of your site and the needs of your users. Remember that many Web users are comfortable with the default underlining, and that color alone may not be enough to differentiate links from the rest of your text.

Using the :hover Pseudo-Class

The :hover pseudo-class lets you apply a style that appears when the user points to an element with a pointing device. This is a useful navigation aid to add to the <a> element, with the result that the link appears highlighted when the user points to it with the mouse. The following style rule shows the :hover pseudo-class with the <a> element as the selector.

```
a:hover {background-color: yellow;}
```

This style rule changes the background color of the link to yellow, which is an effective highlight color. Figure 4-15 shows the results of this style rule.

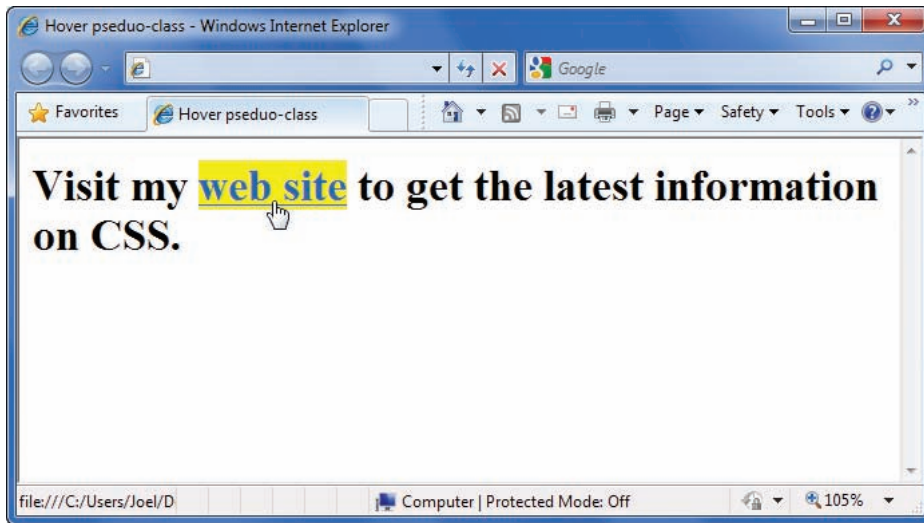


Figure 4-15 The `:hover` pseudo-class is activated by the mouse pointer

Using the `:first-letter` Pseudo-Element

Use the `:first-letter` pseudo-element to apply style rules to the first letter of any element. This lets you create interesting text effects, such as initial capitals and drop capitals, which are usually set in a bolder and larger font. Initial capitals share the same baseline as the rest of the text, while drop capitals extend down two or more lines below the text baseline. To apply `:first-letter` to build an initial capital, specify a style rule like the following:

```
p:first-letter {
  font-weight: bold;
  font-size: 200%;
}
```

This creates a first letter that is bold and twice the size of the `<p>` font. For example, if the `<p>` element has a font size of 12 points, the initial cap will be 24 points.

To make sure that this rule applies only to one paragraph, rather than every paragraph in the document, a class name needs to be added to the style rule. To solve this problem, add a class name, such as *initial*, to the rule, as shown in Figure 4-16.



The `:hover` pseudo-class works on other elements as well; for example, you can use `hover` to highlight any text that a user points to with their mouse pointer.

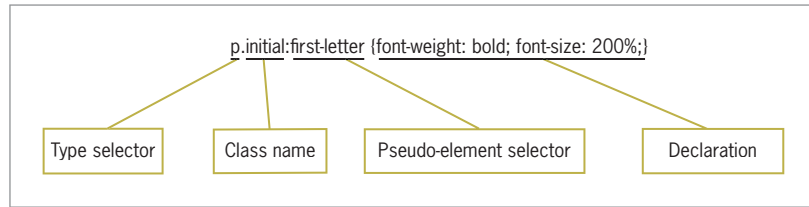


Figure 4-16 Using a class selector with a :first-letter pseudo-element

This style rule affects only `<p>` elements with the class value of *initial*, as shown in the following code:

```
<p class="initial">From the far north they heard a low wail of the wind, and Uncle Henry and Dorothy could see where the long grass bowed in waves before the coming storm. There now came a sharp whistling in the air from the south, and as they turned their eyes that way they saw ripples in the grass coming from that direction also.</p>
```

Figure 4-17 shows the result of the style rule.

Initial capital

From the far north they heard a low wail of the wind, and Uncle Henry and Dorothy could see where the long grass bowed in waves before the coming storm. There now came a sharp whistling in the air from the south, and as they turned their eyes that way they saw ripples in the grass coming from that direction also.

Figure 4-17 Initial capital styled with the :first-letter pseudo-element

You can make the initial capital a drop capital by adding the `float` property to the rule, which allows the letter to extend downward. The `float` property is described in Chapter 6. Here is a `:first-letter` style rule with the `float` property added:

```
p.dropcap:first-letter {
    font-weight: bold;
    font-size: 200%;
    float: left;
}
```

Notice that the class has been changed to signify that this first letter is a drop capital. Remember, you can set the class attribute to any naming value that makes sense to you.

This style rule affects only `<p>` elements with the class value of *dropcap*, as shown in the following code:

```
<p class="dropcap">From the far north they heard a low wail of the wind, and Uncle Henry and Dorothy could see where the long grass bowed in waves before the coming
```

storm. There now came a sharp whistling in the air from the south, and as they turned their eyes that way they saw ripples in the grass coming from that direction also.</p>

Figure 4-18 shows the result of the new style rule.

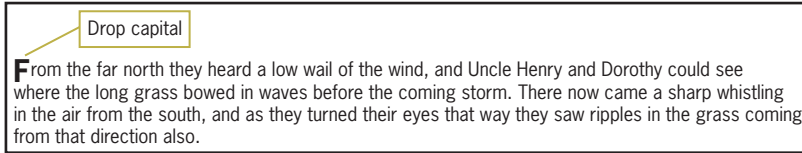


Figure 4-18 Drop capital using the `:first-letter` pseudo-element

The `:first-letter` pseudo-element can only be applied to a block-level element. In addition, only the following properties can be applied to the `:first-letter` selector:

- Font properties
- Color properties
- Background properties
- Margin properties
- Padding properties
- Word-spacing
- Letter-spacing
- Text-decoration
- Vertical-align
- Text-transform
- Line-height
- Text-shadow
- Clear

Using the `:first-line` Pseudo-Element

The `:first-line` pseudo-element works in much the same way as `:first-letter`, except for the obvious difference that it affects the first line of text in an element. For example, the following rule sets the first line of every `<p>` element to uppercase letters:

```
p:first-line {text-transform: uppercase;}
```

The problem with this code is that it affects every `<p>` element in the document. As you saw in the preceding `:first-letter` selector,

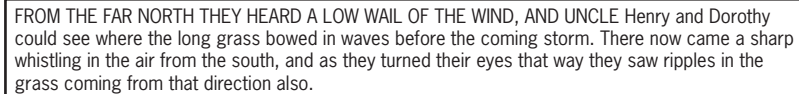
you can add a class attribute to more narrowly define the application of the `:first-line` style:

```
p.introduction:first-line {text-transform: uppercase;}
```

This rule transforms to uppercase the first line of the `<p>` element that contains the following code:

```
<p class="introduction">From the far north they heard a low wail of the wind, and Uncle Henry and Dorothy could see where the long grass bowed in waves before the coming storm. There now came a sharp whistling in the air from the south, and as they turned their eyes that way they saw ripples in the grass coming from that direction also.</p>
```

Figure 4-19 shows the results of the style rule.



FROM THE FAR NORTH THEY HEARD A LOW WAIL OF THE WIND, AND UNCLE Henry and Dorothy could see where the long grass bowed in waves before the coming storm. There now came a sharp whistling in the air from the south, and as they turned their eyes that way they saw ripples in the grass coming from that direction also.

Figure 4-19 First line transformation using the `:first-line` pseudo-element

The `:first-line` pseudo-element can only be applied to a block-level element. In addition, only the following properties can be applied to `:first-line`. Notice that `:first-line` does not support padding, margin, or border properties:

- Font properties
- Color properties
- Background properties
- Word-spacing
- Letter-spacing
- Text-decoration
- Text-transform
- Line-height
- Text-shadow
- Clear

Using the `:before` and `:after` Pseudo-Elements

The `:before` and `:after` pseudo-elements let you insert content in your Web page that is created by the style sheet rather than contained in your document text. Your Web page content may include

terms that must be used repeatedly. These can be inserted by a style sheet rather than having to enter them over and over in your content. A good example of this is having the word *Figure* in the title of all of your figures. For example, this style rule will insert the word *Figure* followed by a colon before any <p> text that has the class figtitle:

```
p.figtitle:before {content: "Figure: "}
```

The :before pseudo-element places generated content before the selected element, and the :after pseudo-element places generated content after the selected element. Here's an example that uses :before to add generated content to a glossary. Style rules select the elements and apply generated content based on the class name.

```
p.term:before {content: "Term: "; font-weight: bold;}
p.definition:before {content: "Definition: "; font-weight: bold;}
```

These style rules apply to the following HTML code. Figure 4-20 shows the results of the following code.

```
<p class="term">Quasar</p>
<p class="definition"> A quasi-stellar radio source
(quasar) is a very energetic and distant galaxy with
an active galactic nucleus. They are the most luminous
objects in the universe.</p>
```

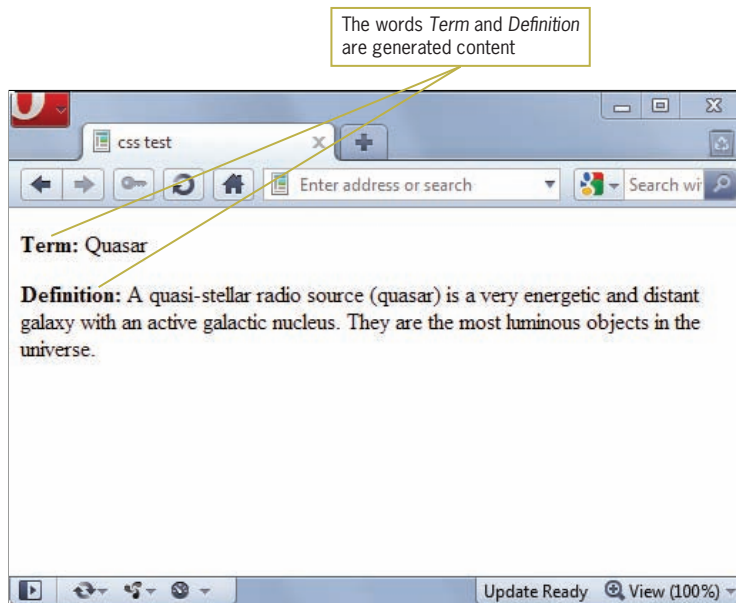


Figure 4-20 Results of using the :before pseudo-element

Understanding How the Cascade Affects Style Rules

One of the fundamental features of CSS is that style sheets **cascade**. This means that multiple style sheets and style rules can apply to the same document. HTML authors can attach a preferred style sheet, while the reader might have a personal style sheet to adjust for preferences such as human or technological handicaps. However, only one rule can apply to an element. The CSS cascading mechanism determines which rules are applied to document elements by assigning a weight to each rule based on the following three variables:

- Specificity of the selector
- Order of the rule in the style sheet
- Use of the !important keyword

Determining Rule Weight by Specificity

Another method of determining style rule weight is the specificity of the rule's element selector. Specificity determines which rule, if there is a conflict, applies to your HTML elements. Rules with more specific selectors take precedence over rules with less specific selectors.

Specificity is actually a complex calculation that is performed by the browser to determine which rule takes precedence, based on adding the weight values of the possible style rules that affect an element.

Specificity conflicts normally occur when multiple style sheets affect a Web page, which is common in larger content management systems, and when compounded selectors are used to narrowly target a specific element on a page. For example, the following style rules target a <p> element. The first applies to all <p> elements on the page, and the second applies to <p> elements that are specifically within a <div> element. The more specific descendant selector determines the outcome, so the <p> elements contained within a <div> will be blue.

```
p {color: red;}  
div p {color: blue;}
```

This is a very simple example of specificity to illustrate the concept. The following general rules make it easier to understand which style rule takes precedence.

- Inline styles, using the style attribute, have the greatest weight.
- Styles with id selectors override styles with class selectors.
- Class selectors override simple type selectors.

Determining Rule Weight by Order

CSS applies weight to a rule based on its order within a style sheet. Rules that are included later in the style sheet order take precedence over earlier rules. Examine the following style rules for an example:

```
body {color: black;}
h1 {color: red;}
h1 {color: green;}
```

In this example, <h1> elements in the document appear green because the last style rule specifies green as the color. This also applies to internal and external style sheets. Internal style sheet rules have greater weight than external style sheet rules

Determining Rule Weight with the !important Keyword

A conflict can arise when multiple rules apply to the same element. By default, rules that are closest to the element, or more specific, always take precedence. The **!important** keyword ensures that a particular rule will always apply. The following style sheet states a rule for <p> elements that sets the font family to arial, regardless of other rules that might apply to the element:

```
<style type="text/css">
p {font-family: arial !important;}
</style>
```

The !important keyword also allows you to specify that a rule should take precedence no matter what the order in the style sheet. Examine the following style rules for an example:

```
body {color: black;}
h1 {color: red !important;}
h1 {color: green;}
```

In this example, <h1> elements in the document will appear red, even though the rule that sets the color to green would normally take precedence because of its order in the document.

However, it is always best to use the !important keyword sparingly since it defeats the flexible nature of CSS.

CSS3 Selectors

CSS3 adds more new selectors that let you narrow down the exact element you want to select with even greater precision. Many of the new selectors let you choose elements based on where they reside in the document structure, letting you make selections such as the first or last paragraph on every page, or setting different

styles for odd or even rows of a table or list. Most browsers support these selectors, but make sure to test carefully for compatibility. In this section, you will learn about the more commonly used CSS3 selectors. You will find a complete list in Appendix A.



You can view CSS3 selector compatibility charts at:
www.css3.info/modules/selector-compat.

CSS3 includes a number of new types of selectors, including the following:

- Substring matching attribute
- Structural pseudo-class
- UI element states

Substring Matching Attribute Selectors

These selectors match parts of an attribute value, such as the beginning or ending of a string of text. Table 4-3 lists the syntax and description for substring matching attribute selectors.

Selector Syntax	Description
p[att^="val"]	Matches any p element whose att attribute value begins with "val"
p[att\$="val"]	Matches any p element whose att attribute value ends with "val"
p[att*="val"]	Matches any p element whose att attribute value contains the substring "val"

Table 4-3 Substring Matching Attribute Selectors

For example, the following rule selects div elements whose class attribute begins with the word *content*:

```
div[class^="content"]
```

Structural Pseudo-Class Selectors

Earlier in this chapter you read how an HTML document is a tree structure of parent and child elements. The structural pseudo-classes let you select elements based on where they reside in the document tree. Table 4-4 lists the syntax and description for structural pseudo-class selectors.

Selector Syntax	Description
:root	Matches the document's root element; in HTML, the root element is always the HTML element
p:nth-child(n)	Matches any <p> element that is the <i>n</i> th child of its parent
p:nth-last-child(n)	Matches any <p> element that is the <i>n</i> th child of its parent, counting from the last child
p:nth-of-type(n)	Matches any <p> element that is the <i>n</i> th sibling of its type
p:nth-last-of-type(n)	Matches any <p> element that is the <i>n</i> th sibling of its type, counting from the last sibling
p:last-child	Matches any <p> element that is the last child of its parent
p:first-of-type	Matches any <p> element that is the first sibling of its type
p:last-of-type	Matches any <p> element that is the last sibling of its type
p:only-child	Matches any <p> element that is the only child of its parent
p:only-of-type	Matches any <p> element that is the only sibling of its type
p:empty	Matches any <p> element that has no children (including text nodes)

Table 4-4 Structural Pseudo-Class Selectors

For example, if you wanted to style the last <p> element on a Web page, you can use this style rule:

```
p:last-child {border-bottom: solid thin black;}
```

UI Element States Selectors

The UI element states selectors let you choose an element based on its state of user interaction. Table 4-5 lists the syntax and description of UI element states selectors.

Selector Syntax	Description
:enable	Matches any interface element, such as a form control, that is in an enabled state
:disabled	Matches any interface element, such as a form control, that is in a disabled state
:checked	Matches any option button or check box that is in a selected state

Table 4-5 UI Element States Selectors

For example, you could highlight a selected option button with the following style rule:

```
input:checked {background-color: yellow;}
```

Chapter Summary

This chapter presents the basic syntax of the CSS language. You learned about the different methods of selecting elements and applying style rules in a variety of ways. You saw that the CSS basic selection techniques are often powerful enough to handle most document styling. You learned that the class and id attributes let you create naming conventions for styles that are meaningful to your organization or information type. As you will see in the upcoming chapters, CSS is an easy-to-use style language that lets you gain visual control over the display of your Web content.

- CSS rules can be combined with your HTML code in a number of ways. CSS rules are easy to write and read.
- CSS uses inheritance and cascading to determine which style rules take precedence.
- Basic style rules let you apply style rules based on standard element selectors. You can combine the selectors and declarations to create more powerful style expressions. You can also select elements based on the contextual relationship of elements in the document tree.
- You can use the class and id attribute selectors, which are often paired with the <div> and HTML elements. You can create rules, name them, and apply them to any element you choose.
- The pseudo-class and pseudo-element selectors let you change the color and styling of hypertext links and the effect elements of a document, such as first line and first letter, that are not signified with the standard HTML elements.

Key Terms

important—A CSS keyword that lets the user override the author's style setting for a particular element.

cascade—Style sheets originate from three sources: the author, the user, and the browser. The cascading feature of CSS lets these multiple style sheets and style rules interact in the same document.

child element—An HTML element contained within another element.

declaration—The declaration portion of a style rule consists of a property name and value. The browser applies the declaration to the selected element.

inheritance—The order of CSS rules dictating that child elements inherit rules from parent elements.

<link> element—An HTML element that lets you establish document relationships, such as linking to an external style sheet.

parent element—An HTML element that contains child elements.

property—A quality or characteristic stated in a style rule, such as color, font-size, or margin. The property is a part of the style rule declaration.

pseudo-class—An element that selects elements based on characteristics other than their element name.

pseudo-element—An element that lets you change other aspects of a document that are not classified by elements, such as applying style rules to the first letter or first line of a paragraph.

recommendation—The final stage of development by the W3C, indicating that a technology release has been reviewed and tested extensively.

selector—The part of a style rule that determines which HTML element to match. Style rules are applied to any element in the document that matches the selector.

style rule—The basic unit of expression in CSS. A style rule is composed of two parts: a selector and a declaration. The style rule expresses the style information for an element.

style sheet—A set of style rules that describes a document's display characteristics. There are two types of style sheets: internal and external.

type selector—A CSS selector that applies a rule to every instance of the element in this document.

universal selector—A selector that lets you quickly select groups of elements and apply a style rule.

value—The precise specification of a property in a style rule, based on the allowable values for the property.

Review Questions and Exercises

1. What are the two parts of a style rule?
2. What are the three ways to combine CSS rules with your HTML code?
3. List two reasons to state a style using the style attribute.
4. What are the advantages of using an external style sheet?
5. What is the inheritance default for CSS rules?
6. What is the benefit of the !important declaration?
7. Write a basic style rule that selects <h1> elements and sets the color property to red.
8. Add the <p> element as an additional selector to the rule you created for Question 7.
9. Add a font-size property to the rule and set the size to 14 points.
10. Write a style rule that selects elements only when they appear within <p> elements and set the color property to red.
11. Write the style rule for a class selector named note. Set the font-weight property to bold.
12. Restrict the rule you developed for Question 11 so it can be used only with <p> elements.
13. What is the difference between <div> and ?
14. Write a style rule that sets the default document text color to red.
15. What is the advantage of working with the class attribute?
16. What element does this selector choose?

p ul li

17. What element does this selector choose?
`div p *`
18. What element does this selector choose?
`p.warning`
19. What is the advantage of working with the id attribute?
20. Write a style rule that applies a yellow background color to `<a>` elements when the user points the mouse to a hypertext link.

Hands-On Projects

1. Visit the World Wide Web Consortium Web site and find the CSS Techniques for Web Content Accessibility Guidelines (www.w3.org/TR/WCAG10-CSS-TECHS). Write an essay describing the benefits of using CSS for accessibility and the primary CSS features you would use to ensure accessible content on the Web.
2. By yourself or with a partner, choose a mainstream publishing Web site, such as a newspaper or periodical site. Examine the style characteristics of the site. What common styles can be applied across the site, such as headings, paragraphs, and bylines? Write an analysis of the site's style requirements, and list the styles you would include in the site's style sheet.
3. In this project, you will have a chance to test a few simple style rules on a standard HTML document and view the results in your browser.
 - a. Using your HTML editor, create a simple HTML file (or open an existing file) that contains `<body>`, `<h1>`, and `<p>` elements. Save the file as **csstest1.htm** in the Chapter04 folder in your work folder.
 - b. Add a `<style>` element to the `<head>` section, as shown in the following code.

```
<head>
<title>CSS Test Document</title>
<style type="text/css">

</style>
</head>
```

- c. Add a style rule that uses *body* as a selector and sets the color property to green, as shown in the following code:

```
<style type="text/css">
body {color: green;}
</style>
```

- d. Save **csstest1.htm** and view it in the browser. All of the document text should now be green.

- e. Now add a style rule that sets <h1> elements to be displayed in black:

```
<style type="text/css">
body {color: green;}
h1 {color: black;}
</style>
```

- f. Save **csstest1.htm** and view the results in the browser.

- g. Finally, add a style rule that sets a margin for <p> elements to 30 pixels:

```
<style type="text/css">
body {color: green;}
h1 {color: black;}
p {margin: 30px;}
</style>
```

- h. Save **csstest1.htm** and view the results in the browser.

4. In this project, you will have a chance to test a few basic selection techniques on a standard HTML document and view the results in your browser. Save the file and view it in your browser after completing each step.

- a. Using your HTML editor, create a simple HTML file (or open an existing file) that contains <body>, <h1>, <p> elements, and so on. Save the file in the Chapter04 folder in your work folder as **csstest2.htm**.

- b. Add a <style> element to the <head> section, as shown in the following code:

```
<head>
<title>CSS Test Document</title>
<style type="text/css">

</style>
</head>
```

- c. Write a style rule that uses *body* as a selector and sets the color property to the color of your choice.
 - d. Find two elements on the page, such as `<h1>` and `<h2>`, which can share the same characteristics. Write a single style rule that applies to both elements. Set the color property to red and the margin property to 20px.
 - e. Find one element that contains another, such as a `` or `<q>` element within a `<p>` element. Write a descendant selector rule that affects the contained element and sets the color property to green.
5. In this project, you will have a chance to test a few advanced selection techniques on a standard HTML document and view the results in your browser. Save the file and view it in your browser after completing each step.
- a. Using your HTML editor, create a simple HTML file (or open an existing file) that contains the elements: `<body>`, `<h1>`, `<p>`, and so on. Save the file in the Chapter04 folder in your work folder as **csstest3.htm**.
 - b. Add a `<style>` element to the `<head>` section, as shown in Exercise 4.
 - c. Write a rule for a class selector named *heading*. Set the color property to red and the font-size property to 200%. Apply the heading class to an `<h1>` or `<h2>` element in the document.
 - d. Write a rule for a class selector named *emphasis*. Set the color property to blue and the background-color property to yellow. In the document, add a `` element to a span of words that you want to highlight. Apply the emphasis class to the `` element.

Individual Case Project

Revisit your project proposal and the site specifications you created in Chapter 3. How will you implement Cascading Style Sheets into your project Web site? In the next few chapters, you will learn how to control typography, white space, borders, colors, and backgrounds with CSS. Think about each of these style characteristics

and how you will apply them to your page designs. In addition, make a list of possible class names you might use to identify your content. For example, consider using class names for the following page characteristics, as well as creating some of your own:

- Body copy
- Header (possibly different levels)
- Footer

Team Case Project

Revisit your project proposal and the site specifications you created in Chapter 3. Each team member is responsible for individual templates, such as the home page template, the section page template, and the article level page template.

In the next few chapters, you will learn how to control typography, white space, borders, colors, and backgrounds with CSS. Decide how you will handle these style characteristics in your Web site. Each team member should create a suggested list of styles and naming conventions for use in your site.

For example, you might have top-level, secondary-level, and tertiary-level headings. What names will you use for consistency throughout the site? You might want to name these *A-head*, *B-head*, and so on. Also, you will need to segregate and name the different copy styles in your site. For example, you might have a different body copy style for the main page than you have on a secondary section or on reading-level pages. Think about the different style properties that you will be able to manipulate (you can look these up in Appendix B) and how you will consistently use, manage, and name these style characteristics for your project Web site. Meet as a team to review each member's ideas and come to an agreement on the proposed styles and naming conventions.