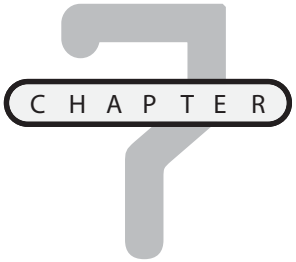
Part III

Cascading Style Sheets



An Introduction to Cascading Style Sheets

Up to this point in the book, you have relied on the browser to determine how to present the content that makes up your (X)HTML documents. In this chapter you will learn how to exercise greater control over how browsers render document content through the application of Cascading Style Sheets (CSS). CSS allows you to separate presentation from content, making both stronger as a result. You can use it to do things like change foreground and background colors and specify font type and color. You can use it to make content visible and invisible, control content placement, and to move things around the browser window. In short, CSS allows detailed control over the way your content in displayed within your visitor's browsers.

Specifically, you will learn:

- The basics of CSS syntax
- The different ways that CSS can be integrated into your web documents
- About CSS specificity and how CSS's cascading rules are applied
- How to use CSS to modify the presentation of text
- How to use CSS to modify color and backgrounds

Project Preview: The Rock, Paper, Scissors Game

In this chapter's web project, you will learn how to create a new web game named Rock, Paper, Scissors. This game pits the player against the computer as both attempt to outguess one another. As Figure 7.1 shows, the player enters moves by clicking on one of two graphic buttons.

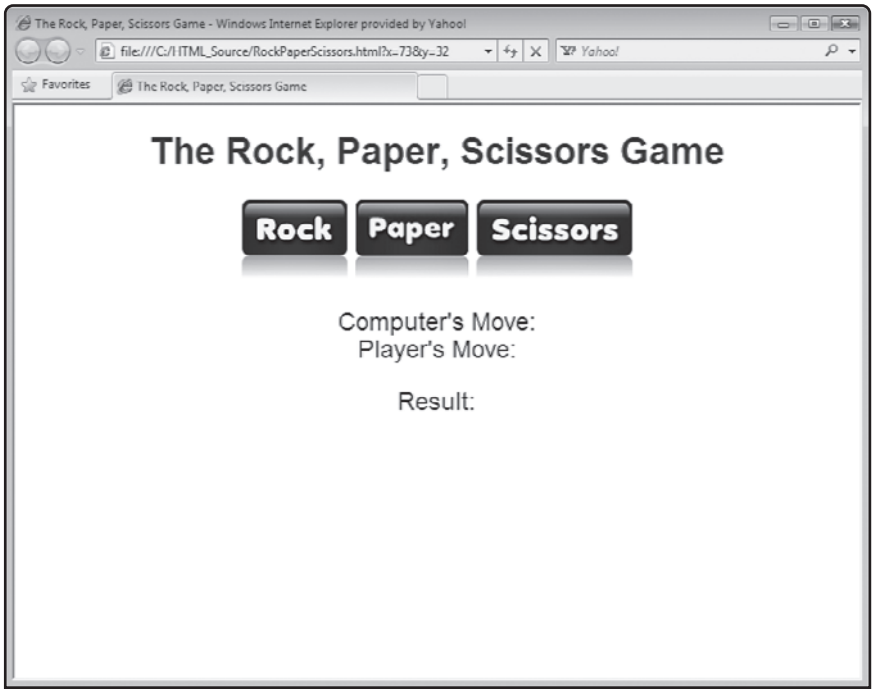


Figure 7.1
The Rock, Paper, Scissors game when first loaded into the browser.

As soon as the player selects a move, the game generates a random move on behalf of the computer. The game analyzes the player and computer's guesses to determine the results of the game. The winner is determined based on the following rules.

- Rock crushes scissors
- Paper covers rock
- Scissors cut paper
- Matching moves result in a tie

Figure 7.2 shows the result of a typical round of play. As you can see, in this example the player's move of Rock has beaten the computer's move of Scissors.

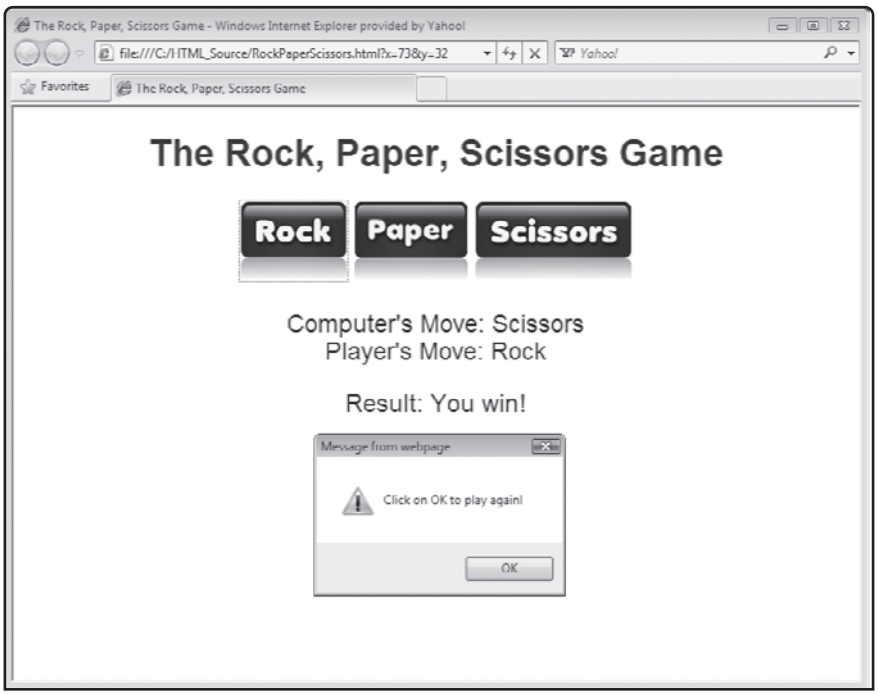


FIGURE 7.2
A popup dialog window is displayed at the end of each round of play.

A popup dialog window is displayed along with the game's results. To play again, the player must click on this OK button. Rather than rely on the browser to determine how to best render the game's content, this application makes use of an external style sheet that specifies font size and color of its heading and paragraph elements and manages the placement of the game's graphics.

INTRODUCING CSS

Cascading Style Sheets or *CSS* is a stylesheet programming language used to describe the way that content written in a markup language is presented. Web developers use CSS to control and influence the presentation of HTML and (X)HTML documents. However, CSS can be used in conjunction with any kind of XML document. CSS provides numerous advantages, including:

- The separation of content and presentation
- The ability to apply style consistently across your web pages
- Reduction of network bandwidth, resulting in pages that load and render more quickly

Prior to the introduction of CSS, (X)HTML presentation could only be influenced using attributes that were added to (X)HTML elements. Unfortunately, this meant intermixing

content and presentation, making web documents longer, more complex, and more difficult to maintain. The arrival of CSS provided web developers with the ability to separate content from presentation, simplifying the development and maintenance of both, which of course, resulted in both aspects getting stronger. Instead of having to repeatedly embed presentation attributes in elements strewn repeatedly through web pages, CSS allowed web developers to develop style sheets and to apply the style rules in those sheets to one or more web pages. Thanks to CSS, you could specify and control the presentation of all of the web pages for an entire website using a single style sheet if you wanted. You could later give the entire website a face-lift by modifying only its style sheet. Needless to say, CSS rocks!

As you will learn in this chapter, CSS includes a prioritization scheme that determines how style rules are applied in the event one or more of them matches the same element. As a result of these prioritization rules, CSS rules *cascade* downward to document elements in a predictable manner. CSS rules can be created to control presentation aspects like font type, size, and color as well as background styles, borders, and the content alignment. You can use it to change the way your tables and forms look and to control the presentation of graphics.

The CSS 1.0 specification was published in December 1996. It got off to a somewhat slow start before finally working its way into mainstream web development. CSS 2.0 was published in May 1998 but has yet to be fully supported by modern web browsers. CSS 3.0 is currently under development. As is the case with HTML and XHTML, the W3C is responsible for the ongoing development of CSS.

Due to lack of universal browser support for CSS 2.0, this book's focus in on CSS 1.0 and it is strongly recommended that you do so as well.

CSS is an integral part of web development. As such, both this chapter and the next are dedicated to explaining and demonstrating how to work with it. However, CSS is its own language, with its own unique syntax and worthy of coverage in its own book. Complete coverage of every CSS property and their range of values exceed the space available in this book. Instead, this book's focus is on providing you with a good foundation with sufficient examples to get you well on your way to mastering CSS. To view the full range of CSS properties, visit http://www.w3schools.com/CSS/css_reference.asp, as shown in Figure 7.3.

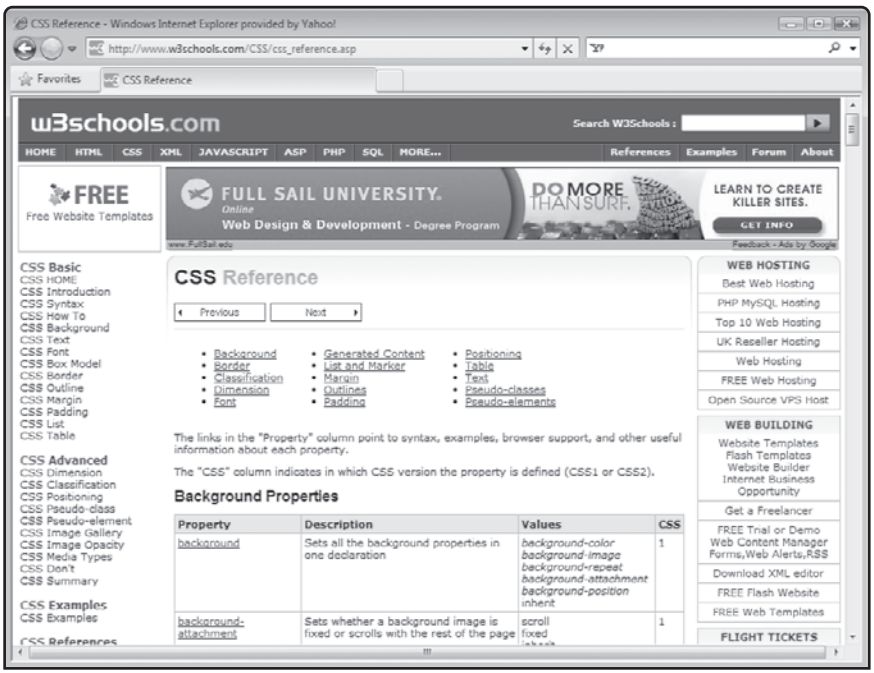


FIGURE 7.3
A complete CSS reference is available online.

Understanding the Basics of CSS Syntax

CSS has a simple syntax. It uses English keywords to specify the styles and their values. There are several ways that CSS can be used, including inline styles, internal style sheets, and external style sheets. This chapter will show you how to work with all three options. CSS style sheets consist of lists of *rules*. CSS rules are roughly analogous to (X)HTML elements. As demonstrated in Figure 7.4, CSS rules are made up of selectors and a declaration block.

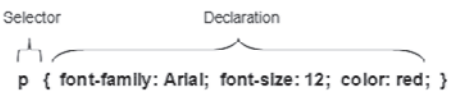


FIGURE 7.4
A CSS rule can have one or more selectors and a declaration block.

Style rules can be written in order to affect the presentation of a specific (X)HTML element or multiple elements. Rules specify the elements they affect through the selector. *Selectors* are used to identify the (X)HTML elements to which style rules are applied. You can specify more than one selector in a rule as long as you separate each one by a comma. The *declaration block* is made up of one or more declarations, allowing you to manipulate more than one aspect of an element's presentation at a time. *Declarations* are embedded inside an opening

{ character and a closing } character. These brace characters are roughly analogous to (X) HTML's ≤ and ≥ characters. Declarations are made up of one or more property/value pairs, as outlined in Figure 7.5.

FIGURE 7.5
Declarations are made up of one or more property/value pairs.

Properties identify the presentation aspect of an element that a rule is modifying. Each declaration property is separated from its corresponding value by a colon. A *value* is a setting that is applied to the specified element(s). In addition, each property/value pair is separated from other property/value pairs by a semicolon. If a declaration has only one property/value pair, you have the option of omitting the semicolon. In addition, you can also leave off the semicolon from the last property/value pair that makes up a declaration. Some properties accept multiple values separated from each other by blank spaces.

Crafting Rule Selectors

In order to effectively use CSS to control the presentation of content on your web pages, you have to understand the concept of specificity. Specificity is a term used to identify the scope that a CSS style rule has within a document (e.g., how many elements the rule affects). When you define a CSS style rule, its selector defines its specificity. CSS supports a wide range of selectors, each with different levels of specificity.

Universal

A universal selector is a selector that matches every element found in a web document. To define a universal selector, you must use the * (wild card) character, as demonstrated here:

```
* {color: red;}
```

When executed, this rule will display all text on the web document in red. Rules defined using a universal selector have the least amount of specificity.

Element

An element selector is one that matches all instances of a given element within a web document, as demonstrated here:

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

This rule will display the text of all paragraphs on the web document in blue. Although element selectors have greater specificity than universal selectors, they are still not very specific.

Class

A class selector is one that matches any elements assigned a specified class name attribute. As discussed in Chapter 2, any number of elements on a web document can be assigned to the same class. Though only moderately specific, the class selector is more specific than the element class.

```
.total {color: green;}
```

Note that to specify a class as a selector, you must precede the class name with a . character.

Pseudo Class

A pseudo class selector is used to match elements that exist in a specific state. In terms of specificity, pseudo class selectors are roughly equivalent to class selectors. Only five types of pseudo classes are supported. These classes are:

- :link
- :visited
- :active
- :hover
- :focus

As you can see, all pseudo classes begin with the : character followed by one of five keywords that represent the status of document links. The following rule demonstrates how to work with pseudo class selectors.

```
:hover {color: purple;}
```

This CSS style rule instructs the browser to display in purple any link on the web page when the mouse pointer is hovered over it.

ID

An ID selector is one that matches a unique element based on its assigned attribute ID. Since each element's ID, when assigned, must be unique within a web document, an ID selector is very specific. To reference an ID in a selector, you must precede the ID with the # character, as demonstrated here:

```
#score {color: blue;}
```

When processed, this CSS style rule will display in blue text the element that has been assigned an ID attribute of `score`.

Specifying More Complex Selectors

In addition to specifying individual selectors, CSS lets you create a number of more granular selectors by combining, grouping, and supporting descendant selectors. As demonstrated here, two or more selectors can be combined to create a more complex selector.

```
h1.main {color: blue;}
```

Here, a selector has been defined that matches all level 1 headers that are assigned to the `.main` class.

You can also create more complex selectors by adding together two or more comma-separated selectors, as demonstrated here:

```
h1, h2, h3 {color: green;}
```

Here, all level 1, 2, and 3 headings will be displayed as green text.

Another type of complex selector, referred to as a descendant selector, can be defined by using two or more selectors separated by blank spaces. This type of selector matches up against elements that have a specified contextual relationship to one another, as demonstrated here:

```
.score strong {color: red;}
```

Here, any strong elements located within an element assigned to the `score` class are displayed in red.

Integrating CSS into Your HTML Pages

To use CSS, you need a means of integrating CSS into your web documents. (X)HTML provides several different ways of doing so, as outlined here:

- **Inline Styles.** Styles embedded within (X)HTML element tags.
- **Embedded Style Sheets.** Style rules embedded within the `head` section of your (X)HTML pages.
- **External Style Sheets.** Style rules stored in external files and linked back to your (X)HTML pages.

Using Inline Styles

To use inline styles, you embed CSS styles inside your (X)HTML element tags. To do so, add an optional `style` attribute to each element whose presentation you want to modify. Inline styles

are not constructed as CSS rules. They do not make use of selectors. Instead, only declaration blocks are included in inline styles. For example, the following statements add an inline style to a paragraph tag, instructing the browser to display the color of the element's text in `green`.

```
<p style = "color: green";>Hello World!<p>
```

If needed, you can include as many property/value pairs in your declaration as you want. Just remember to keep each one separated by a semicolon, as demonstrated here:

```
<p style = "font-size: 9; color: red; text-align: center;">Hello World!</p>
```

As a general rule, you will want to avoid using inline styles and to defer to using external style sheets or perhaps embedded style sheets. Inline styles require that you intermix presentation with your (X)HTML markup. This eliminates one of CSS's primary advantages: the ability to separate presentation from content and structure. Another challenge in working with inline styles is that since they are applied only to individual elements, you must repeatedly use them on similar types of elements to ensure that your web page has a consistent presentation. This eliminates another advantage of CSS, the ability to present from a single location. Worse still, what if you later change your mind about how things work and want to give your (X)HTML documents a face-lift? You'd have to revisit every element in every page to make those changes. Smart web developers avoid this mistake, only using inline elements in very specific situations when it is necessary to ensure that a particular element's presentation is fine-tuned.

Managing Individual Documents with Embedded Style Sheets

While working with inline styles may be okay for a small (X)HTML document, it is not a good idea to use it with documents whose presentation requires a lot of detailed customization. For this type of situation, a much better option is the use of embedded style sheets. Embedded style sheets help to make it easier to apply a consistent look and feel to your web document by enabling you to globally apply style rules to all matching elements.

To add an embedded style sheet to a web page, you must work with the `style` element. The `style` element is a block-level element that must be placed in a document's `head` section, which helps to keep presentation separate from document structure. Any CSS rules placed within the `style` element are then globally applied to any matching elements found throughout the document.

As an example of the effects of adding an embedded style sheet to an (X)HTML document, look at the following example.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">

  <head>
    <meta http-equiv="Content-type" content="text/xhtml; charset=UTF-8" />
    <title>Chapter 7 - Sample Web Page</title>
  </head>

  <body>
    <h1>This heading should be purple, underlined, and centered</h1>
    <p>This paragraph should be bold, italic, and blue.</p>
    <h2>This heading should be orange on a black background and centered</h2>
    <p id="p1">This paragraph should be red, bold, and italic.</p>

  </body>

</html>
```

As you can see, this document consists only of markup and a little content and relies on the browser to handle the document's presentation. Every web browser has its own built-in default browser style sheet. Figure 7.6 shows the resulting web page that is rendered when this document is loaded into the browser.

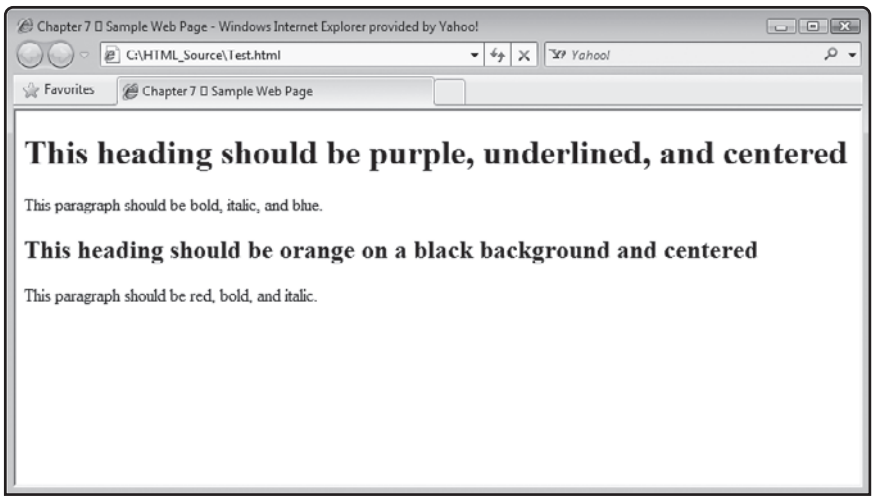


FIGURE 7.6
An example of a small web document.

To spruce things up a bit, let's modify the document by adding an embedded style sheet, as demonstrated in the following example.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">


<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">

  <head>
    <meta http-equiv="Content-type" content="text/xhtml; charset=UTF-8" />
    <title>Chapter 7 - Working with an Embedded Style Sheet</title>

    <style type = "text/css">

      /*This rule formats all level 1 headings*/
      h1 {
        color: purple;
        text-decoration: underline;
        text-align: center;
      }

      /*This rule formats all level 2 headings*/
      h2 {
        color: orange;
        background-color: black;
        text-align: center;
      }

      /*This rule formats all paragraphs*/
      p {
        font-weight: bold;
        font-style: italic;
        color: blue;
      }

      /*This rule formats a paragraph whose id = p1*/
      #p1 {
```

```
        color: red;
      }

    </style>

  </head>

  <body>
    <h1>This heading should be purple, underlined, and centered</h1>
    <p>This paragraph should be bold, italic, and blue.</p>
    <h2>This heading should be orange on a black background and centered</h2>
    <p id="p1">This paragraph should be red, bold, and italic.</p>

  </body>

</html>
```

As you can see, an embedded style sheet has been added to the document that is made up of four separate CSS style rules. The first rule modifies the appearance of all `h1` elements, displaying the color of their text in purple, making it underlined and centered. The second rule modifies the appearance of all `h2` elements and displays their text in orange on a black background that is centered. The third rule governs the presentation of paragraphs, displaying them in bold, italic, and blue text. The fourth rule affects a specific paragraph on the page. This paragraph is identified by an assigned ID attribute of `p1` (CSS denotes IDs by preappending a # character to them).

The fourth rule demonstrates CSS's ability to cascade overlapping rules, by modifying the presentation of one specific paragraph tag. The color property assignment in the fourth rule conflicts with the color property assignment of the third rule. CSS resolves this situation by allowing the most specific selector's color property assignment to override the change made by the third rule. In CSS, rules with greater specificity take precedent.

Note the inclusion of a comment located just before each style rule in the previous example. As you can see, CSS comments begin with the /* characters and end with the */ characters.

Figure 7.7 demonstrates the effect that the addition of the embedded style sheet has had on the web document's appearance once rendered by the web browser.

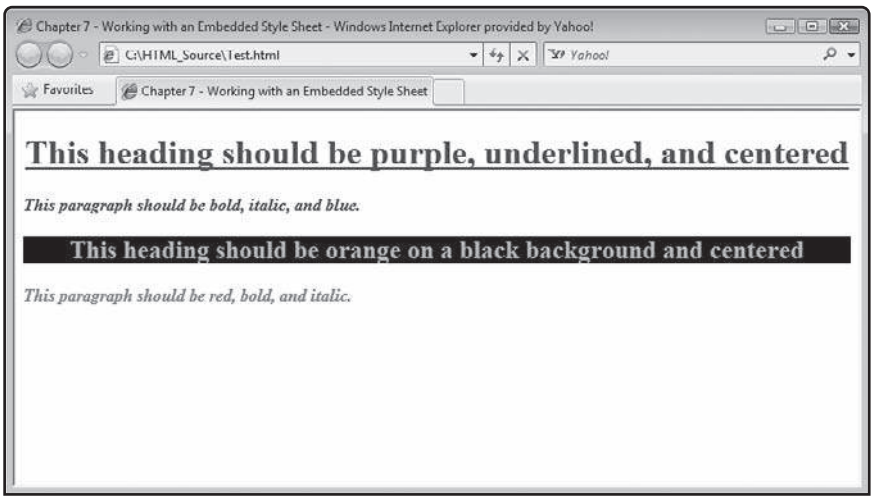


FIGURE 7.7

Examples of how an embedded style sheet can be used to control the presentation of text.

When formulating style sheets you can format your CSS rules any way you want as long as you follow CSS basic syntax. This means you are free to add blank space and carriage returns to help improve readability. For example, you can format your CSS rules in a compact manner, as demonstrated here:

```
h1{color:purple;text-decoration:underline;text-align:center;}
```

Alternatively, you can open things up a bit and make your rules easier to understand by being generous with white space, as demonstrated here:

```
h1 {color: purple; text-decoration: underline; text-align: center;}
```

If your style rules are rather long, you may find it easier to write them in an extended format, as demonstrated here:

```
h1 {
   color: purple;
   text-decoration: underline;
   text-align: center;
}
```

All three of the preceding examples, though laid out differently, are functionally identical. Which format you choose to work with is up to you.

Leveraging the Power of External Style Sheets

The obvious limitation of an embedded style sheet is that you are limited to using it in a single document. To apply the same set of style rules to multiple documents using embedded style sheets, you must copy and paste the embedded style sheets into each document. If you later

want to make a change to the presentation of your pages, you will have to revisit each document and make the same update over and over again. The answer to this problem is external style sheets.

An external style sheet is a plain text file made up of CSS style rules. It is arguably your best option for applying CSS to your (X)HTML documents because it provides all of CSS's primary advantages: the separation of content and presentation, the ability to apply style across multiple web pages, and reduced use of network bandwidth. By using external style sheets you can significantly reduce the overall size and complexity of web documents.

Another way of using external style sheets in your web document is to add an `@import` statement inside a style element, as demonstrated here:

```
<style type = "text/css">
  @import "text.css"
  .
  .
  .
</style>
```

In this example, an external style sheet named text.css has been imported into an embedded style sheet. When imported in this manner, the rules in the external style sheet are processed before any of the rules in the embedded style sheet.

Using the previous example as a starting point, you can easily convert an embedded style sheet into an external style sheet. All you have to do is cut and paste the style rules from the `<style>` element into an external text file (don't forget to also remove the opening `<style>` and closing `</style>` tags from the `head` section of the (X)HTML document).

You can name your external style sheets anything you want as long as you assign them a .css file extension. For example, the following CSS file was created by extracting rules from the previous (X)HTML document. It has been named style.css.

```
h1 {
   color: purple;
   text-decoration: underline;
   text-align: center;
}
h2 {
   color: orange;
   background-color: black;
   text-align: center;
```

```
}
p {
  font-weight: bold;
  font-style: italic;
  color: blue;
}
#p1 {
  color: red;
}
```

Once the `style` element has been removed from the (X)HTML document, the document has become substantially smaller as shown below. As a result, the document is easier to read, understand, and modify.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">

  <head>
    <meta http-equiv="Content-type" content="text/xhtml; charset=UTF-8" />
    <title>Chapter 7 - Working with an External Style Sheet</title>
    <link href = "style.css" type = "text/css" rel = "stylesheet" />
  </head>

  <body>
    <h1>This heading should be purple, underlined, and centered</h1>
    <p>This paragraph should be bold, italic, and blue.</p>
    <h2>This heading should be orange on a black background and centered</h2>
    <p id="p1">This paragraph should be red, bold, and italic.</p>
  </body>

</html>
```

To connect the new external style sheet to the (X)HTML document, a `link` element has been added to the `head` section. This element contains three elements, explained next.

- **href.** Specifies the URL of the external style sheet.
- **type.** Identifies the MIME type of the document that is being linked.
- **rel.** Specifies the relationship of the linked file to the (X)HTML document.

When the web browser loads this web document, it automatically downloads and applies all of the CSS style rules found in the external style sheet to the document. Even better, the browser automatically caches the style sheet in memory. As a result, if the user loads another one of your web pages that uses the same external style sheet, the style sheet won't have to be downloaded again. As a result, things will occur more quickly, making for happier users. No wonder external style sheets are preferred by most web developers.

You are not limited to working with a single style sheet in your web document. If you want to use more than one external style sheet, place a link to each one in the `head` section. You might, for example, have a style sheet that you link to all of the documents that make up your website and then apply a second more specific style sheet to a given set of pages in order to further customize the presentation of any unique content they might have.

You can also include more than one embedded style sheet in your documents, placing each within its own `style` elements. In fact, while I would not recommend it, you can use inline styles, embedded style sheets, and external style sheets all within a single document.

Understanding How CSS Rules Are Applied

CSS style rules can often conflict with and overlap one another. To ensure that these conflicts are resolved in a predictable and orderly manner, CSS makes use of two resolution techniques, specificity and cascading. Specificity dictates that more specific selectors override less specific selectors. Cascading means that when two selectors of equal specificity conflict, the style rules occurring later override those that occur earlier.

Specificity

Because selectors can be both specific and general, it does not take much for them to come into conflict with one another. When that happens, you must look at the specificity of each selector in order to determine which one will take precedence. While specificity conflicts may be easy to avoid on small web documents, they tend to occur with greater frequency on larger and more complicated documents. Specificity is one of the most difficult concepts to understand when it comes to CSS. In CSS, different weights are assigned to selectors based on their specificity. When you have a web page with elements whose presentation is not being rendered as expected, it is usually because of problems with specificity. A good understanding of CSS specificity rules is therefore critical to web developers.

In CSS, more specific selectors override less specific selectors. In order for you to be able to predict and understand how conflicting style rules will be applied, you need to understand the relative amount of specificity that is assigned to different types of selectors, as outlined in the following list.

- Universal selectors are not specific.
- Element selectors are more specific than universal selectors.
- Class selectors are more specific than element selectors.
- Pseudo class selectors have the same level of specificity as call selectors.
- ID selectors are more specific than class and pseudo selectors.
- Inline styles have a higher specificity than ID selectors.

In addition to understanding how specificity applies to different types of selectors, you also need to understand the following basic rules, which further govern how conflicts are resolved.

Selectors with more specificity override selectors with less specificity. If selectors have equal specificity, the last one wins. The number of selectors in a rule is also cumulative. So a rule with more selectors has a greater specificity than a rule with few selectors of the same type. However, a rule with more selectors of lesser type cannot outweigh a rule with a selector of a higher type (e.g., 5 element selectors do not provide more weight than a single class selector, and a group of class selectors cannot outweigh a single ID selector).

As an example of how all this works, look at a couple of examples, starting with the following rules.

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

While you might instinctively think that after processing these rules the browser would display all paragraph text in blue, it does not. Instead, paragraph text is displayed in green because the first selector is regarded as being more specific since it specifies more elements than the second rule. Next, consider the following example.

```
h1 {color: green;}
.firstheading {color: red;}
```

In addition to these rules, let's assume that you have the following element in the web document.

```
<h1 class = ".firstheading">Welcome!<h1>
```

What do you think happens here? Well, the color of all headings in the document would be green except for the heading assigned to the `.firstheading` class. Why? Because class selectors have a greater specificity than element selectors.

Cascading

Okay, so what happens when two rules with equal specificity come into conflict, as demonstrated in the following example?

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

Since the level of specificity is the same, the browser turns to cascading to determine the result. As a result, the last specified rule cascades over top of the previous rule and is the rule that is applied.

CSS also uses cascading to help determine how style rules should be applied when multiple style sheets are used. For example, a document may have links to two external style sheets. When this occurs, CSS applies rules in each style sheet in the order their associated `link` elements are listed. So in the case of a tie, the style rules on the second external style sheet will override those of the first external style sheet.

If multiple embedded style sheets are present, they are processed in the order in which they are defined, with later embedded style sheets overriding previous ones. Embedded style sheets take precedence over external style sheets and both of these style sheets take precedence over the browser's built-in style sheet. Modern browsers also allow users to configure their own user style sheet. If present, user style sheets take precedence over the browser style sheet. However, embedded and external style sheets take precedence over both user and browser style sheets. If this is not confusing enough, if your web documents make use of any inline styles, they take precedence over all style sheets.

What to Do When All Else Fails

Sometimes trying to figure out why a given piece of content is not being presented the way you want it to be can be very difficult, especially with big web documents with large or multiple style sheets. If you run into a situation where you are unable to work out precisely what is going on, you can exercise a little extra muscle by adding the `!important` keyword to the CSS style rule you want given extra preference.

Note that use of the `!important` keyword is generally discouraged and should only be used in exceptional situations where you have run out of ideas for remedying the situation any other way. The following example demonstrates the use of this keyword.

```
p {color : green !important;}
```

Note that only two things override a CSS rule that uses the `!important` keyword. First is the use of the keyword in a conflicting rule that has greater specificity over which occurs later

in the cascade order. The second is the use of the `!important` keyword in a user style sheet, which always takes precedence.

Styling Fonts and Color

You have seen numerous examples of number style rules in this chapter. Most have focused on the presentation of text color or fonts. Let's spend a little time digging deeper into CSS font, text color, and background properties so that you will better understand what you have seen.

Influencing Font Presentation

CSS provides you with control over the selection and appearance of fonts. Using different CSS properties, you can specify font type, size, and a number of other attributes. Table 7.1 provides a list of CSS properties that you can use to control the selection of fonts and to configure their size, width, style, and weight.

Table 7.1 CSS Font Properties

Property	Description
font-family	A prioritized list of font types, such as Arial and Verdana, that specify the font to be used. The list of fonts must be separated by commas. The first available font on the user's computer is automatically used.
font-size	Specifies the size of the font.
font-stretch	Expands or condenses a font's width. Available options include: `normal`, `wider`, and `narrower`.
font-style	Specifies how the font should be displayed. Available options include: `normal`, `italic`, and `oblique`.
font-weight	Specifies font boldness. Available options include: `normal`, `bold`, `bolder`, and `lighter`.

The following example demonstrates how to specify font selection, font style, and font size for various document elements.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">
```

```
  <head>
    <meta http-equiv="Content-type" content="text/xhtml; charset=UTF-8" />
    <title>Chapter 7 - Working with an External Style Sheet</title>
    <link href = "style.css" type = "text/css" rel = "stylesheet" />
  </head>

  <body>

    <h1>This heading is displayed in the Arial font using italics</h1>
    <p>This paragraph is displayed in the Garamond font at 150% default
size.</p>

  </body>

</html>
```

As you can see, this XHTML page's content includes a heading and a paragraph element. A link located in the `head` section configures the document's presentation using an external style sheet named style.css. The contents of this style sheet are shown here:

```
h1 {
  font-family: Arial;
  font-style: italic;
}

p {
  font-family: Garamond;
  font-size: 150%;
}
```

As you can see, the first style rule selects the Arial font (`font-family: Arial;`) for any level 1 headings. This font family is universally available on virtually every type of computer. The second declaration in the first rule assigns a value of `italic` to the `font-style` property, so that the browser will display any level 1 headings in italic. The second rule shown above affects the presentation of any paragraphs, setting the `font-family` property to `Garamond` and the `font-size` property to `150%`.

Figure 7.8 shows how this example looks once it has been loaded into the web browser.

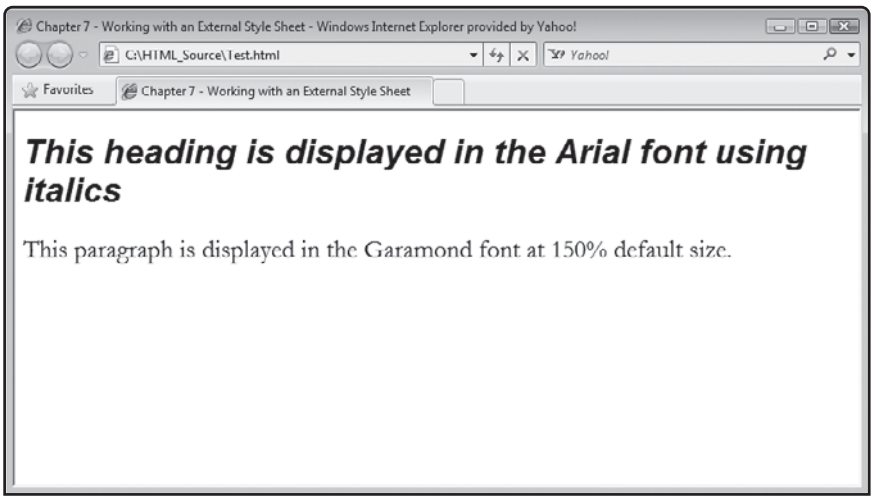


FIGURE 7.8
Examples of how to use CSS to modify different properties.

Take note of both the `font-family` and `font-size` properties used in this example style sheet. Both merit additional discussion. The `font-family` property lets you specify a list of one or more font families as part of a CSS style rule. A *font family* control consists of a collection of font definitions for a given font type. These definitions include different-sized fonts. For example, the Arial font family typically consists of font sizes that range from 8 to 72 points.

When you specify a lot of font families using the `font-family` property, you are really outlining your preferred font type. If the first font family in the list is found on the user's computer, it will be used. If it is not found, the browser will look for the next font in the list. If none of the fonts in the family is found, the browser will default back to its default font type.

CSS supports five generic font families. If you want, you can specify these font families as part of the list in your style rules. These generic font families provide you with a fallback mechanism in the event none of the other font families you specify in your style rules are found on your visitors' computers. When used, the browser will locate a font type on the user's computer that fits the characteristics of the font family, allowing you at least minimal influence of the presentation of text.

- **Cursive.** Fonts in this font family typically have joined strokes and other presentation characteristics that resemble cursive writing. Examples of compatible fonts include Sanvito, Caflisch Script, Corsiva, and Ex Ponto.
- **Fantasy.** Fonts in this family are highly decorative and ornamental. Examples of compatible fonts include Cottonwood, Critter, and Studz.
- **Monospace.** Fonts in this family are proportionately spaced, much like the font used by typewriters. Examples of compatible fonts include Courier, Prostige, and Everson Mona.

- **San Serif.** Fonts in this family have decorative finishes like flaring and cross strike. Examples of compatible fonts include MS Tohoma, MS Arial, Helvetica, and MS Verdana.
- **Serif.** Fonts in this family are among the most decorative and ornamental. They have distinguished finishing strikes and/or tapered endings. Examples of fonts in this family include Times New Roman, MS Georgia, Garamond, and Bodoni.

You can use the `font-family` property in any style rule that works with fonts (e.g., headings, paragraphs, etc.). CSS allows for the downward inheritance of font property assignment. In most cases, property assignments will be inherited by any elements embedded within the element where the property assignment is made. Therefore, it is often desirable to specify a rule that sets font properties for the `body` element, and to allow these property assignment values to be inherited by the rest of the elements in the `body` section, as demonstrated here.

```
body {font-family: "Times New Roman", Georgia, Serif;}
```

Note that the Times New Roman entry in the previous example was enclosed in quotation marks. This is required for any multi-word font names that include spaces.

When it comes to the use of the `font-size` property, CSS supports a number of different ways of specifying measurements, which you can use when specifying CSS properties. The full range of measurement options available are listed in Table 7.2.

TABLE 7.2 CSS MEASUREMENT UNITS

Unit	Description
%	Percentage
cm	Centimeter
em	One em is equal to the current font size. Two em is two times the size of the current font, etc.
ex	One ex is equivalent to the x-height of a font (which is approximately half the current font size)
in	Inch
mm	Millimeter
pc	Pica (one pc is equivalent to 12 points)
pt	Point (one pt is equivalent to 1/72 inch)
px	Pixels (the smallest addressable area on a computer screen)

Controlling the Presentation of Text

In addition to providing you with the ability to specify different font properties, CSS also allows you to modify properties that affect the presentation of text content. Table 7.3 lists style properties that provide control over presentation features like height, letter spacing, and indentation.

TABLE 7.3 CSS TEXT FORMATTING PROPERTIES

Property	Description
text-align	Sets text alignment. Available options include `left`, `right`, `center`, and `justify`.
text-indent	Indents the first line of text.
text-decoration	Applies a decoration to text. Available options include `none`, `underline`, `overline`, `blink`, and `line-through`.
line-height	Specifies the distance between lines.
letter-spacing	Specifies the amount of space between characters.
word-spacing	Specifies the amount of space between words.

As an example of how to work with various CSS text-formatting properties, look at the following example.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">

  <head>
    <meta http-equiv="Content-type" content="text/xhtml; charset=UTF-8" />
    <title>Chapter 7 - Working with an External Style Sheet</title>
    <link href = "style.css" type = "text/css" rel = "stylesheet" />
  </head>

  <body>

    <h1>Centered Heading</h1>
    <p id = "p1">Right justified paragraph (underlined)</p>
    <p id = "p2">Left justified paragraph</p>
```

```
  </body>

</html>
```

Here, a document has been created that displays a heading and two paragraphs. The first paragraph has been assigned an ID of `p1` and the second paragraph has been assigned an ID of p2. Note that an external style sheet named style.css has been referenced using a `link` statement located in the document's `head` section. The contents of that style sheet are shown here:

```
h1 {
  text-align: center;
}

#p1 {
  text-decoration: underline;
  text-align: right;
}

#p2 {
  text-align: left;
}
```

When displayed, this example produces the results shown in Figure 7.9.

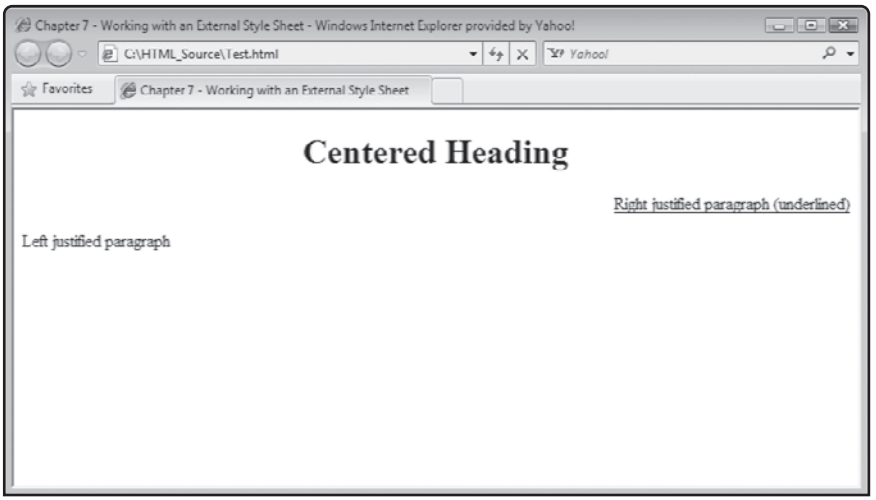


FIGURE 7.9
Examples of how to use CSS to control text alignment.

Specifying Foreground and Background Properties

CSS also allows you to specify the colors and backgrounds that are displayed on your web pages. Table 7.4 lists CSS style properties you can use to control presentation features like text color and window background color.

TABLE 7.4 CSS COLOR AND BACKGROUND PROPERTIES

Property	Description
color	Specifies the color to be used as the foreground color.
background-image	Specifies the URL of an image file to be used as the background.
background-color	Specifies the color to be used as the background color.
background-repeat	Specifies whether the background image should be tiled. Available options include `no-repeat`, `repeat-x`, and `repeat-y`.
background-position	Specifies the starting position for the background. Available options include `center`, `top`, `bottom`, `right`, and `left`.

CSS provides a number of different ways of specifying color values. You can specify a color using its color name (blue, green, purple, etc). If you prefer, you can specify a color using its hexadecimal value (example: #FFFFFF equals white, #000000 equals black, #FF0000 equals red). Alternatively, you can determine color using the JavaScript `rgb()`, in which you just specify three numbers in the range of 1 to 255, representing different red, green, and blue values (example: `rgb(255, 255, 255)` equals white, `rgb(0, 0, 0)` equals black, and `rgb(255, 0, 0)` equals red). Lastly, you can specify color using the `rgb()` function along with percentages of red, green, and blue `rgb(r%, g%, b%)`.

The following example demonstrates how to work with a number of the properties listed in Table 7.4.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">

  <head>
    <meta http-equiv="Content-type" content="text/xhtml; charset=UTF-8" />
    <title>Chapter 7 - Working with an External Style Sheet</title>
    <link href = "style.css" type = "text/css" rel = "stylesheet" />
```

```
  </head>

  <body>
    <h1>All text on this page should appear in green</h1>
    <p>The background color for this page is yellow.</p>
  </body>

</html>
```

As you can see, this document links to a style sheet named style.css, which contains the following CSS style rule.

```
body {
  color: green;
  background-color: yellow;
}
```

This style rule configures all text displayed in the web pages as green on a yellow background. Note that the `color` property assignment here has been made to the `body` section. However, through a process referred to as inheritance, this property assignment flows down to elements declared within the document's body. Figure 7.10 shows an example of what you will see when you load this example into the web browser.

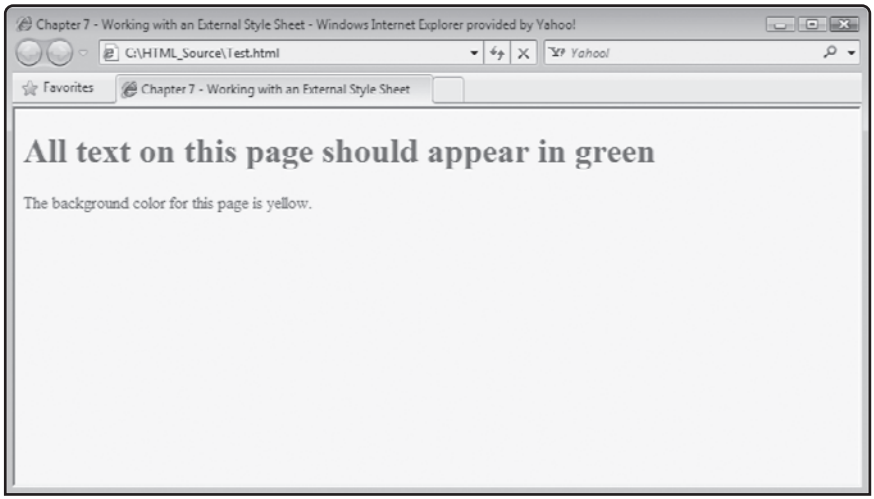


FIGURE 7.10
Examples of how to use CSS to modify a web page's text and background colors.

Validating CSS Syntax

Just as is the case with (X)HTML, the W3C provides free access to a CSS validation service located at http://jigsaw.w3.org/css-validator/, as shown in Figure 7.11. Use this site to ensure that your style sheets are well formed and syntactically valid. Failure to create a valid style sheet can render one or more of its rules completely ineffective and may even result in the entire style sheet failing, leaving your (X)HTML page's presentation to the mercy of your visitor's browser.

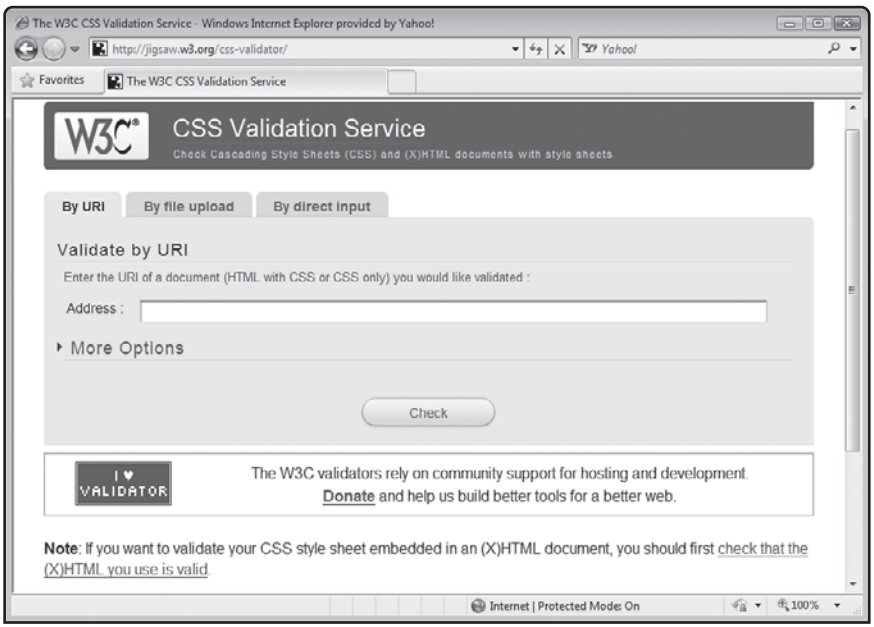


Figure 7.11
Use the free CSS style sheet analyzer provided by W3C to ensure all your style sheets are valid and well formed.

Back to the Rock, Paper, Scissors Game

All right, now it is time to return your attention to this chapter's project, the Rock, Paper, Scissors game. This game will be created using a combination of XHTML, JavaScript, CSS, and the DOM. Together these four technologies can be used to create interactive web pages that dynamically update the display of web content. Collectively, these four technologies are sometimes referred to as *DHTML*, which stands for *Dynamic HTML*. As you'll learn by following along with this project, DHTML helps to take web development to the next level by turning static web pages into an interactive experience.

Designing the Application

This web project is more complicated than any of the previous projects you have worked on so far. To help make things easier to digest, the development of this web application will be broken down into a series of seven steps, as outlined here:

1. Create a new XHTML document.
2. Develop the document's markup.
3. Add `meta` and `title` elements.
4. Create the document's script.
5. Specify document content.
6. Create an external style sheet.
7. Load and test Rock, Paper, Scissors.

Step 1: Creating a New XHTML Document

Let's begin the development of this game project by creating a new web document. Do so using your preferred code or text editor. Save the document as a plain text file named RockPaperScissors.html. This web document will make use of CSS style rules. Therefore, you will need to create a second file named rps.css.

Step 2: Developing the Document's Markup

The next step in the development of the Rock, Paper, Scissors game is to assemble the web document's markup. To do so, add the following elements to the RockPaperScissors.html file.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">

  <head>

  </head>

  <body>

  </body>

</html>
```

Step 3: Adding meta and title Elements

Next, let's ensure that the `head` section is well formed by adding the following elements to it.

```
<meta http-equiv="Content-type" content="text/xhtml; charset=UTF-8" />
<title>The Rock, Paper, Scissors Game</title>
```

This web application is made up of both a web document and an external style sheet named rps.css. As we are working on the document's `head` section, let's go ahead and set up the link to the external style sheet by adding the following statement to the end of the `head` section.

```
<link href = "rps.css" type = "text/css" rel = "stylesheet" />
```

Step 4: Specifying Document Content

Now it is time to define a form that the user will use to interact with the game. This form will display three graphic controls labeled `Rock`, `Paper`, and `Scissors`. The player will click on these buttons when making moves during game play. To add the form to the document, add the following statements to the `body` section.

```
<h1>The Rock, Paper, Scissors Game</h1>

<form action = "RockPaperScissors.html">

  <div>
   <input type = "image" src = "rock.png" onClick = play("Rock") />
   <input type = "image" src = "paper.png" onClick = play("Paper") />
   <input type = "image" src = "scissors.png" onClick = play("Scissors") />
  </div>

</form>
```

Note that the form consists of three `input` elements embedded within a `div` element and that the `form` element's `action` attribute has been set to the web page itself and not to a server-side form handler. Also, note that the form has no Submit button (and thus no need for a form handler).

Each of the `<input>` tags displays a different graphic, representing the game's graphic button controls. Each `input` element also ends with a `javascript` statement that uses the `onClick()` event handler to execute a JavaScript function, which you will create in a minute, passing a text string representing the player's move.

The game will dynamically display text that shows the player and computer's moves as well as a message showing who won each time a new round is played. To facilitate the display of this information, add the following elements to the end of the document's `body` section.

```
<p>
  Computer's Move: <span id="computer"> </span> <br />
  Player's Move: <span id="player"> </span>
</p>

<p>
  Result: <span id="result"> </span>
</p>
```

As you can see, two paragraphs containing three `span` elements have been used to develop a template through which text can be dynamically displayed. Each span element has been assigned a unique ID, allowing the application's JavaScript statements to dynamically update content using the DOM.

Step 5: Creating the Document's Script

Now that you have added all of the document's markup, it is time to create the JavaScript responsible for controlling the operation of the game. Let's begin by adding the following statements to the document's `head` section.

```
<script type = "text/javascript">
<!-- Start hiding JavaScript statements

// End hiding JavaScript statements -->
</script>
```

These statements provide the markup needed to support the definition of the script. The JavaScript itself is just one large function named `playerMove()`. It will be called to execute whenever the player clicks on one of the game's graphic button controls. To create this function, embed the following statements inside the `script` element's opening `<script>` tag and closing `</script>` tags:

```
function play(playerMove) {

  randomNo = 1 + Math.random() * 2; //Generate a random number

                                    //from 1 - 3
```

```
randomNo = Math.round(randomNo);  //Change number to an integer

//Assign the computer's move based on the random number
if (randomNo == 1) computerMove = "Rock"
if (randomNo == 2) computerMove = "Paper"
if (randomNo == 3) computerMove = "Scissors"

//Compare the computer's and the player's move
switch (computerMove) {
  case "Rock":
    if (playerMove == "Rock") {
      document.getElementById('result').innerHTML = "You tie!"
    }
    if (playerMove == "Paper") {
      document.getElementById('result').innerHTML = "You win!"
    }
    if (playerMove == "Scissors") {
      document.getElementById('result').innerHTML = "You lose!"
    }
    break;
  case "Paper":
    if (playerMove == "Rock") {
      document.getElementById('result').innerHTML = "You lose!"
    }
    if (playerMove == "Paper") {
      document.getElementById('result').innerHTML = "You tie!"
    }
    if (playerMove == "Scissors") {
      document.getElementById('result').innerHTML = "You win!"
    }
    break;
  case "Scissors":
    if (playerMove == "Rock") {
      document.getElementById('result').innerHTML = "You win!"
    }
    if (playerMove == "Paper") {
      document.getElementById('result').innerHTML = "You lose!"
    }
```

```
      if (playerMove == "Scissors") {
        document.getElementById('result').innerHTML = "You tie!"
      }
     break;
  }

  document.getElementById('computer').innerHTML = computerMove
  document.getElementById('player').innerHTML = playerMove

  window.alert("Click on OK to play again!");

}
```

This script is executed when the player clicks on one of the game's graphic controls, causing the control's `onClick` event to trigger and pass a text string representing the selected button control. The function definition includes a parameter named `playerMove`. The text string argument passed as input to the script is mapped to this parameter, assigning the text string as the value of the parameter, which acts as a variable. The function compares the player's move to the computer's move, automatically generated by the script on behalf of the computer, to determine the result.

The first two statements in the script generate a random number from 1 to 3, representing the computer's move. The random number is assigned to a variable named `randomNo`. The next three statements assign a text string to a variable named `computerMove`. The value assigned depends on the value of the random number. A value of `1` results in an assignment of `Rock`. A value of `2` results in the assignment of `Paper`, and a value of `3` results in the assignment of `Scissors` as the text string representing the computer's move.

Now that both the player and the computer's moves are known, the script needs to compare their values in order to determine a result. This is accomplished using a `switch` code block that contains a number of embedded `if` statement code blocks.

The first `case` statement executes if the computer's assigned move is `Rock`. It is followed by three `if` statement code blocks. Only one `if` statement code block executes, depending on the value of the player's move. Lastly, a `break` statement is executed, terminating the execution of the rest of the `switch` code block. In similar fashion, the second and third `case` statements execute when the computer's assigned move is `paper` or `scissors`.

At this point, the script has determined whether the player has won, lost, or tied. What happens next is a little complicated and is what causes the script to dynamically update the web page, without requiring the page to be refreshed or reloaded. In simple terms, depending on

the result of the script's analysis of the player and computer's moves, a statement like the one shown here is executed.

```
document.getElementById('result').innerHTML = "You tie!"
```

This script statement merits some explanation. It requires some understanding of the DOM, as was covered in Chapter 1. It begins by referring to the document object, which is always the top-most document in the DOM tree. Next, the `getElementById()` method is executed. This method retrieves a reference to a document element based on its assigned ID, which is passed to the method as a string argument. Lastly, the `innerHTML` property is used to assist a text string as the document element's new value, dynamically updating it on the fly. Note that the dot (`.`) character is used to glue together the different parts of this statement, referred to as dot notation. The last three statements use similar logic to dynamically update the `span` elements responsible for displaying the player and computer's moves and then display a popup dialog window that instructs the player to reload the web page in order to play again.

Step 6: Creating an External Style Sheet

The RockPaperScissors.html document uses an external style sheet to configure the presentation of its content. This style sheet is saved in a file named rps.css. The rules stored in this style sheet are shown here:

```
h1 {
     color: MidnightBlue;
     text-align: center;
     font-family: Arial;
     font-size: 36px;
   }

div {
     text-align: center;
    }

p { text-align: center;
    font-family: Arial;
    font-size: 24px;
  }
```

The first rule in the style sheet applies to all of the `h1` headings in RockPaperScissors.html. It will display them using a color of `MidnightBlue`, centering the text using the `Arial` font using a font size that is 36 pixels high. The second rule applies to the document's `div` element, which

contains the three graphic button controls. The third and final style rule specifies the presentation of the document's paragraphs, centering them and displaying them using an `Arial` font that is 24 pixels high.

Step 7: Loading and Testing the Rock, Paper, Scissors Game

At this point, your copy of the RockPaperScissors.html document should be complete. To help ensure you've assembled it correctly, take a look at the following example, which shows a complete copy of the finished document.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">

  <head>
    <meta http-equiv="Content-type" content="text/xhtml; charset=UTF-8" />
    <title>The Rock, Paper, Scissors Game</title>
    <link href = "rps.css" type = "text/css" rel = "stylesheet" />

    <script type = "text/javascript">
    <!-- Start hiding JavaScript statements

      function play(playerMove) {

        randomNo = 1 + Math.random() * 2; //Generate a random number
                                          //from 1 - 3

        randomNo = Math.round(randomNo);  //Change number to an integer

        //Assign the computer's move based on the random number
        if (randomNo == 1) computerMove = "Rock"
        if (randomNo == 2) computerMove = "Paper"
        if (randomNo == 3) computerMove = "Scissors"

        //Compare the computer's and the player's move
        switch (computerMove) {
          case "Rock":
            if (playerMove == "Rock") {
```

```
      document.getElementById('result').innerHTML = "You tie!"
    }
    if (playerMove == "Paper") {
      document.getElementById('result').innerHTML = "You win!"
    }
    if (playerMove == "Scissors") {
      document.getElementById('result').innerHTML = "You lose!"
    }
    break;
  case "Paper":
    if (playerMove == "Rock") {
      document.getElementById('result').innerHTML = "You lose!"
    }
    if (playerMove == "Paper") {
      document.getElementById('result').innerHTML = "You tie!"
    }
    if (playerMove == "Scissors") {
      document.getElementById('result').innerHTML = "You win!"
    }
    break;
  case "Scissors":
    if (playerMove == "Rock") {
      document.getElementById('result').innerHTML = "You win!"
    }
    if (playerMove == "Paper") {
      document.getElementById('result').innerHTML = "You lose!"
    }
    if (playerMove == "Scissors") {
      document.getElementById('result').innerHTML = "You tie!"
    }
    break;
}

document.getElementById('computer').innerHTML = computerMove
document.getElementById('player').innerHTML = playerMove

window.alert("Click on OK to play again!");
```

```
      }

    // End hiding JavaScript statements -->
    </script>

  </head>

  <body>

    <h1>The Rock, Paper, Scissors Game</h1>

    <form action = "RockPaperScissors.html">

      <div>
        <input type = "image" src = "rock.png" onClick = play("Rock") />
        <input type = "image" src = "paper.png" onClick = play("Paper") />
        <input type = "image" src = "scissors.png"
          onClick = play("Scissors") />
      </div>

    </form>

    <p>
      Computer's Move: <span id="computer"> </span> <br />
      Player's Move: <span id="player"> </span>
    </p>

    <p>
      Result: <span id="result"> </span>
    </p>

  </body>

</html>
```

Okay, if you have not done so already, now is the time to load your new web page and to see how your new game works.

A complete copy of the source code for this project, including its style sheet, and the graphics needed to create its graphic controls is available on the book's companion web page, located at www.courseptr.com/downloads.

Summary

This chapter provided an introduction to CSS and explained its role in helping to influence the presentation of content on web pages. You learned the basic syntax required to formulate CSS style rules. This included how to work with inline styles, embedded style sheets, and external style sheets. You also learned how to use CSS to style fonts, text, and foreground and background colors. On top of all this, you learned how to create the Rock, Paper, Scissors game.

Challenges

1. **The Rock, Paper, Scissors game currently makes the assumption that the player knows the rules for playing the game. Rather than rely on this assumption, consider creating another document with detailed instructions for playing the game and add a link to that document on the RockPaperScissors.html document.**
2. **Let's spruce things up a bit by modifying the color of the document's paragraph to something more attractive than the default color of black.**
3. **Lastly, using the `em` element, enclose the `span` element, so that the `player`, `computer`, and `result` values are given additional emphasis when displayed.**