CHAPTER 5

Using Cascading Style Sheets

A cascading style sheet provides a great, flexible way for controlling the look or presentation of your Web pages. Small changes to this one document can dramatically change the appearance of many Web pages. This easy method for controlling the design of your pages allows you to rapidly implement changes to the look and feel of your Web site. Programmers appreciate this single point of control.

A cascading style sheet is a file that ends in the *.CSS* suffix. Developers often name their cascading style sheet *theme.css*. “Theme” is a good choice for the file name, since a style sheet can create the visual theme for a Web site. All you need to do is have all the pages in your Web site reference the same cascading style sheet. It’s also possible to have multiple style sheets all in effect for the same HTML documents. We’ll discuss how that is accomplished later in this chapter.

Creating a Cascading Style Sheet

You can attach a cascading style sheet to your Web page using the **<link>** tag. The code snippet in Figure 5.1 shows an example of such a link. When the HTML page is opened, the browser will search the theme.css style sheet to find rules for handling the content of the page.

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

Figure 5.1: Linking to a cascading style sheet.

You can create the theme.css style sheet using a tool as simple as Notepad, by using a sophisticated tool in a Web design product. Figure 5.2 shows the code for a basic theme.css style sheet.

```
body {font-family : arial,courier new,times new roman;
}

h1 {font-family : times new roman;
}

h3 {font-family : times new roman;
}

P {font-family : courier new;
}
```

Figure 5.2: A basic style sheet for four HTML tags.

Tags

```
element {property:value;
}
```

The styles in the theme.css code in Figure 5.2 are for the HTML **<body>** element, the **<h1>** and **<h3>** heading elements, and the paragraph element. One of the basic things you can do with style sheets is define a default style for each of the HTML elements. As the HTML document is rendered, the styles are read in from the style sheet and applied. There other ways to define styles, such as using style classes, discussed later in this chapter.

When defining the style for an element, the element's tag is listed without the angle brackets (<>) and followed by braces ({}). Within these braces, any available property can be defined. Each property is followed by a colon (:) and then by its associated value. Each value is followed by a semicolon (;). Additional properties may be listed as needed. Once all of the properties are listed, the ending brace is coded.

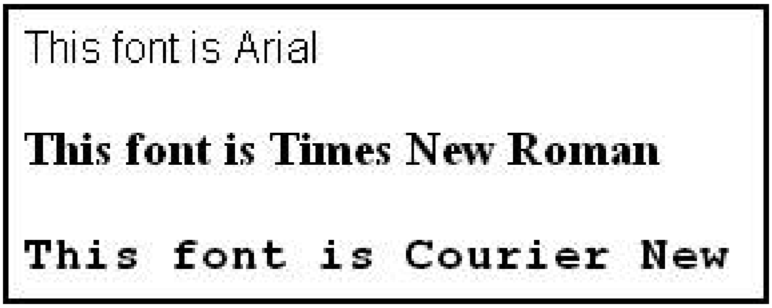
In theme.css, each of the four element styles sets the default font family for that element. In the **body** style, the font family is set to Arial. If Arial is not available on the user's computer, one of the alternate fonts listed will be used. Simply list all of the fonts in order of preference. The first one available on the user's computer will be used. If none of the listed fonts are available, the default font of the user's browser will be used. The **h1** and **h3** styles define the font family as Times New Roman. The fourth style is for the paragraph tag **<p>**. This style sets the default font to Courier New.

A wide variety of fonts are available. *Monospace* fonts such as Courier New use the exact same width for every letter. This makes them very attractive when you need to get text to line up a certain way.

Figure 5.3 shows the code for a sample Web page that links to the theme.css style sheet. The Web page is shown in Figure 5.4.

```
<html>
<head>
<link href="theme.css" rel="stylesheet" type="text/css">
</head>
<body>
This font is Arial
<h1>This font is Times New Roman</h1>
<h3><P>This font is Courier New</P></h3>
</body>
</html>
```

Figure 5.3: The HTML code for a Web page using a style sheet.



This font is Arial

This font is Times New Roman

This font is Courier New

Figure 5.4: Sample font families using a style sheet.

Notice in Figure 5.3 that within the second **<h3>** tag, there is also a **<p>** tag, so two of the styles from themes.css are in effect. In general, the innermost, or *child*, element will inherit properties from a parent element, but the child also has the option to override those properties. Both element styles attempt to change the font family, but the **<p>** tag takes precedence, since it is the innermost tag. Therefore, the text in the **<h3>** tag appears in Courier New instead of Times New Roman.

What Other Properties Do Fonts Have?

In addition to the **font-family** property, you can also set the font size, font style, or font weight. Alternatively, you can set the **font** property, which has several values, including style, variant, weight, size, and family.

Tags

```
{font-family:font1, font2,…

 font-size:size
 font-style:normal
   italic
   oblique
 font-weight:normal
   bold
   bolder
   lighter
   number
 font:style weight
  size family
}
```

Most of the properties have specific options to choose from. For example, **font-weight** can use relative terms such as **lighter**, **normal**, **bold**, or **bolder**. These choices are displayed here in **bold** text. Other properties allow more varied values. These are displayed here in ***bold italics***. For example, **font-weight** accepts a numeric value that is a multiple of 100, from 100 to 900. A value of 100 is the lightest (thinnest) font weight, while 900 is the heaviest (thickest).

The **font-size** property requires a value that may be followed by a code such as **12px**, which means 12 pixels wide. Here are the available size codes:

- **em**, the width of the letter *m* in the current font
- **ex**, the width of the letter *x* in the current font

- **cm**, centimeters
- **mm**, millimeters
- **pc**, picas (1 pica = 4.216 mm)
- **pt**, points (1 point = 1/12 pica)
- **in**, inches
- **px**, pixels

Using the **em** and **ex** sizes allows you to define the size of certain portions of text without needing to know the underlying size of the font. Specifying a value of *.5em*, for example, means you want the font to be 50% smaller than the current text size for the letter *m*. Similarly, to define a size 20% greater than the current text size of the letter *x*, use a value of *1.2ex*. Avoiding hardcoded font sizes provides better support for users who have adjusted their own font sizes.

Use the **font-style** to determine if the text should be printed in its normal form, italics, or as oblique text. (Oblique prints "slanted" text that, to the untrained eye, looks the same as italics.)

There are a number of other text properties not directly associated with a font. These include **color**, **direction**, **line-height**, **letter-spacing**, **text-align**, **text-decoration**,

Tags

```
{color:color
   rgb(r,g,b)
   #rrggbb
 direction:ltr
    rtl
 line-height:normal
      number
      percent
 letter-spacing:normal
      number
 text-align:left
    right
    center
    justify
text-decoration:
    none
    underline
    overline
    line-through
    blink
 text-indent: number
     percent
 text-shadow: color
 horizontal-distance
 vertical-distance
 blur radius
 text-transform:none
      capitalize
      uppercase
      lowercase
 white-space: normal
    pre
    nowrap
 word-spacing:normal
    number
 Unicode-bidi:normal
    embed
    bidi-override
}
```

text-indent, **text-shadow**, **text-transform**, **Unicode-bidi**, **white-space**, and **word-spacing**.

Color may be coded as a specific name, such as *blue* or *red*, as an RGB hex value, or as RGB decimal values (rarely used). Color names are easy to use, but not all browsers present the colors in exactly the same way. The 16 standard HTML colors are shown in Table 5.1, together with their hex values. More colors can be found in appendix E.

Examples of hex values are #000000 (black), #c0c0c0 (silver), and #FFFFFF (white). Each hex code is made up of a pound sign (#) followed by three hex values (00 through FF), representing the red, green, and blue values of the color. Most programmers know that hex is a base-16 number system, where 0=0 and F=15. So, a hex code of #3366CC has 33 for the red value, 66 for the green, and CC for the blue, resulting in a medium grayish-blue.

Table 5.1: Standard Color Names and Hex Codes

Color Name	Hex Code
Aqua	#00FFFF
Black	#000000
Blue	#0000FF
Fuchsia	#FF00FF
Gray	#808080
Green	#008000
Lime	#00FF00
Maroon	#800000
Navy	#000080
Olive	#808000
Purple	#800080
Red	#FF0000
Silver	#C0C0C0
Teal	#008080
Yellow	#FFFF00
White	#FFFFFF

The **direction** property can be set to left-to-right **(ltr)** which is the default, or right-to-left **(rtl)**. **Line-height** can be set to **normal**, a specific number, or a percentage.

Letter-spacing controls the space between text characters. This property can be set to either **normal** or a specific number.

The **text-align** property can be set to **left**, **right**, **center**, or **justify**.

Text-decoration lets you add an underline or overline effect to the text. You can also cause the text to appear with a line through it. This is often used to represent deleted text. A blinking effect can also be added.

The **text-indent** property sets the amount of indention for the first line of text in the paragraph. It can be entered either as a fixed amount or as a percentage.

Text-shadow sets the color for a shadow effect added to the text. Horizontal and vertical distance properties indicate the offset distance of the shadow effect from the text. This property is not supported by many browsers at this time.

Use the **text-transform** option to force the text to appear in uppercase or lowercase, or to capitalize the first letter of every word.

White-space controls the way in which the browser handles the white space within the HTML text. Setting it to **pre** causes the text to be handled as if the HTML **<pre>** tag were specified. Using the **nowrap** value indicates that the text should never wrap down to the next line. It will continue on the same line until the end of the text or a line-break tag (**<br>**).

The **word-spacing** property sets the distance between each word in the text. Set this to a specific size.

If you use the **rtl** directional text, you can set the **Unicode-bidi** property to **bidi-override**, causing the first letter of text to be printed at the right margin, with each following character moving closer to the left margin. Otherwise, **rtl** will simply cause the text to be aligned on the right margin. For an example of this, consider the code in Figure 5.5. If the style sheet in Figure 5.6 is applied, the sentence is printed right to left, as shown in Figure 5.7. The text is aligned on the right margin by default.

```
<p>this text prints right to left</p>
```

Figure 5.5: The HTML code to print a line of text.

```
p {direction:rtl; unicode-bidi:bidi-override;
```

Figure 5.6: The style sheet code that controls the text in Figure 5.5.

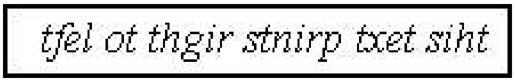

tfel ot thgir stnirp txet siht

Figure 5.7: Right-to-left text.

This example might be a bit silly, as it is rare that you would need to print text right to left. However, most of the other font properties, such as color assignment and weight, don't show up well in printed material.

What Properties Control the Arrangement of an Element?

Use the padding properties to control the spacing around an element.

Tags

```
{padding-bottom:size
       auto
 padding-left: size
    auto
 padding-right: size
    auto
 padding-top: size
    auto
 padding: top right
    bottom left
}
```

Typically, each element on a Web page is assigned a rectangular section or "box." Properties that affect the general positioning and arrangement of content on the page manipulate the format of these boxes. As a general rule, these boxes are not visible. *Padding* refers to the internal space between the content of an element and the element's border. You can control the padding values for each of the four sides of the element, or if you specify the **padding** property, you can set all four at once.

If **padding** contains just a single value, such as *{padding:3px;}*, that value applies to all four sides of the element. If two values are given, such as *{padding:3px 2px;}*, the first value is for the top and bottom, and the second value is for the left and right. When three values are given, the first is for the top, the second for the left and right, and the third for the bottom.

Tags

```
{height:size
  auto
width: size
  auto
 max-height: size
   auto
max-width: size
   auto
 min-height: size
   auto
min-width: size
   auto
}
```

The **height** and **width** properties refer to the size of the element itself. Use a size in one of the formats discussed earlier for the **font-weight** property. The **max-height** and **max-width** properties refer to the maximum size that an element can expand to. Rather than specifying an exact size, these properties set a limit on the element's size. The **min-height** and **min-width** properties control the minimum size the element can be shrunk to.

As the browser integrates a variety of elements on the same page, some will be placed beside others. If necessary, you can force the browser to leave either one side or both sides free of adjacent elements. You would set the **clear** property to **right**, for example, if you wanted to keep the right side clear.

The **bottom** property sets the distance that an element is above the bottom edge of its block area. **Left**, **right**, and **top** properties define the distance the element's content is from the edge of that block.

Use the **float** property to identify how the element should be arranged with other elements. Set it to **right** if it should float to the right, **left** to float left, or **none** to prevent it from floating at all.

Visibility controls whether an element can be seen or not. Set this to **visible** for elements that should be seen, **hidden** for ones that should remain in the background, and **collapse** if the element is to be hidden away from view, but available for quick display if needed.

Overflow determines how the browser handles content that will not fit in the defined space for the object. Set **overflow** to **visible** to guarantee that the content will be visible despite overflowing the element's maximum size. Use **hidden** to cause the overflow content to become invisible. Use **scroll** to indicate that scrollbars should be added to the element to allow access to its entire contents. **Clip** sets the size of the clipped

Tags

```
{clear: none
    both
    left
    right
 bottom:number
   auto
 float: left
   right
   none
 visibility: visible
     hidden
     collapse
 top: number
   auto
 right: number
    auto
 left: number
    auto
 position: static
    relative
    fixed
    absolute
 clip: auto
    rect(top,right,
     left,bottom)
 overflow: visible
    hidden
    scroll
    auto
 vertical-align:
     number
     baseline
     sub
     super
     top
     text-top
     middle
     bottom
     text-bottom
 z-index: auto
     number
}
```

area of an element with overflow. It accepts four values, the top, right, left, and bottom positions, which identify the top right corner of the object, and the bottom left corner of the visible portion of the element.

Use the **position** property to control the way in which the browser places the element on the page. If the property is set to **absolute**, the element's position (top, right, left, and bottom) is relative to the page itself and independent of any parent elements on the page. If the position property is set to **relative**, the position of the element is adjusted from the location at which it would normally appear. So, an element that would normally appear 10 pixels from the top of the page and 20 pixels from the left, with position set to relative, and 5px for both the top and left properties, would appear 15 pixels from the top of the page, and 25 from the left.

Use **z-index** to define layers within the Web page. By default, all of the content is at the "0" index layer. Content placed at z-index 1 will overlay that, and content at z-index 2 will overlay the z-index 1 content. This provides an easy mechanism to overlay content. To avoid layering and keep all content at the same layer as the parent element, specify **auto** as the z-index.

The **vertical-align** property controls the alignment of elements in line with each other within a containing box. **Baseline** is the default value. It causes all the elements in the line to align with each other along the baseline of the containing box. **Sub** and **super** cause elements to align as if they were subscript and superscript, respectively. **Top**, **bottom**, and **middle** cause the elements to align along the top of the highest element, the bottom of the lowest element, or the middle of all the elements. With **text-bottom** or **text-top**, the elements are lined up with the bottom or top of the parent item's **font** property.

What Properties Control the Display of an Element?

The **cursor** and **display** properties provide the ability to customize the look of the cursor and the text of the page. The cursor can be modified to appear as a variety of pointers, arrows, or crosshairs. The **display** property affects text in numerous ways. For example, it can force text to appear in line with other text, or force the text to appear in vertical lists.

Reformatting the mouse pointer may be a useful tool for communicating with users. For example, you could use the cursor to indicate that a certain link provides help information. The **cursor** property sets the appearance of the mouse pointer. Set it to **crosshair** to change the mouse pointer into a targeting crosshair. Use **move** to create a mouse pointer that indicates the element can be moved. This looks similar to the crosshairs, but includes arrows on the end of each line.

The resize values of **cursor** switch the pointer to a sizing arrow that points in the given direction. So, the **e-resize** cursor is an arrow that points east-west, and the **nw-resize** cursor is an arrow that points northwest-southeast.

Use the **text** value of **cursor** to change the mouse pointer to the vertical line commonly used in text areas. **Wait** causes the cursor to change to an hourglass, and **help** changes it to a question mark.

Tags

```
{cursor: auto
    crosshair
    pointer
    default
    move
    e-resize
    ne-resize
    nw-resize
    n-resize
    se-resize
    sw-resize
    s-resize
    w-resize
    text
    wait
    help

 display: none
   inline
   block
   list-item
   run-in
   compact
   marker
   table
   inline-table
   table-row-group
   table-header-group
   table-footer- group
   table-column-group
   table-row
   table-column
   table-cell
   table-caption
}
```

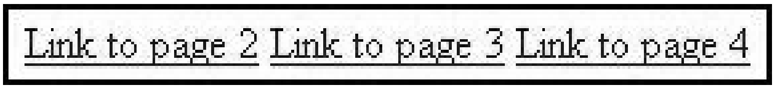
The **display** property has a wide range of options and an even wider range of support from the major browsers. Rather than going over all of these, we'll focus on a few of the more useful options. Set **display** to **none** to cause an element to not be displayed. This is different than the **hidden** value, which was discussed earlier. A hidden element occupies space on the page, and other elements will move as if it were there. **{Display:none}** causes the element to be ignored by the browser, so other elements on the page may be placed in the space that the non-displayed element would have occupied.

The **block** value essentially behaves like paragraphs have always behaved, advancing to a new line and avoiding placing other elements to its right or left. The **inline** option causes an element to display on the current line of the current block.

The code sample in Figure 5.8 creates a series of hypertext links. Figure 5.9 shows how they would normally appear in the browser. The links appear one after another until the right margin is reached, at which point the text moves down to the next line.

```
<a href="abc.html">Link to page 2</a>
<a href="def.html">Link to page 3</a>
<a href="ghi.html">Link to page 4</a>
```

Figure 5.8: HTML code to show three hypertext links.



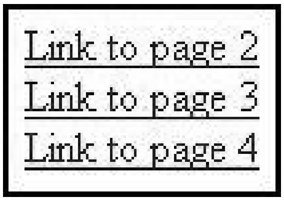
Link to page 2 Link to page 3 Link to page 4

Figure 5.9: Links formatted inline with each other.

If you wanted each link to advance to the next line, similar to the way paragraphs behave, you could use the **display:block** property. To accomplish this, you could add the code in Figure 5.10 to a cascading style sheet. The output would change as shown in Figure 5.11.

```
a {display:block;
}
```

Figure 5.10: A cascading style sheet for block format.



Link to page 2
Link to page 3
Link to page 4

Figure 5.11: Links presented in block format.

Margins define the space between the border and the edge of a containing box.

Tags

```
{margin-bottom:size
    auto
 margin-left: size
    auto
 margin-right: size
    auto
 margin-top: size
    auto
 margin: top right
   bottom left
}
```

Each margin can be set to a specific size, as for the **font-size** property described earlier. The **margin** property lets you set all four margins at once. If only one value is given, all four margins use that value. If two values are given, the top and bottom margins use the first value, and the right and left margins use the second. If three values are given, the first value is for the top margin, the second for the left and right, and the third for the bottom.

Use the **border** property to set the width, style, and color of the border. Set the width to a specific size, as for the **font-size** property. The style can be set to values such as **dashed**, **groove**, **inset**, or **outset**. The default border style is **none**. The color may be coded as a color name, a hex value, or an RGB value.

Tags

```
{border: width
    style
    color
 border-color: color
     rgb
     hex
 border-style: none
     hidden
     dotted
     dashed
     solid
     double
     groove
     ridge
     inset
     outset
 border-width:size
     thin
     medium
     thick
 border-top-color: as above
 border-top-style: as above
 border-top-width: as above
 border-top: as above
 border-bottom-color: as above
 border-bottom-style: as above
 border-bottom-width: as above
 border-bottom: as above
 border-left-color: as above
 border-left-style: as above
 border-left-width: as above
 border-left: as above
 border-right-color: as above
 border-right-style: as above
 border-right-width: as above
 border-right: as above
}
```

By default, no border is shown for an element. Use **border-color** to set just the color property for the border. Similarly, the **border-width** property defines just the width of the element's border. The **border-style** property can be set to **none** to indicate that no border be displayed, or **hidden** to indicate that the border be rendered on the page, but invisibly.

The **dotted**, **dashed**, and **solid** values obviously describe the appearance of the border.

The **groove** border style appears like a channel cut into the surface of the Web page. **Ridge** creates a 3D raised border that surrounds the element. **Double** creates a border within a border. The **inset** border appears sunken into the surface of the Web page, while the **outset** value makes an element appear to be elevated, as if on top of a button.

Use **border-top**, **border-left**, **border-right**, and **border-bottom** to set the border properties for just that section of the element.

To make the links from Figure 5.11 more visually distinct, we could add a border. In this case, the code in Figure 5.12 adds an outset border to make them appear raised above the surface of the Web page, as shown in Figure 5.13.

```
a {display:block; border-style:outset;
    width:14ex;text-align:center;
}
```

Figure 5.12: The style sheet code for anchor tags with special borders.

The code in Figure 5.12 keeps the block attribute from the earlier example and adds the outset border. It also sets the width to **14ex** (fourteen *x*'s) so that the outset borders would not continue all the way to the right margin of the page. The text was also centered within the block so that it lined up nice and neat within the borders. This makes the links look like buttons.

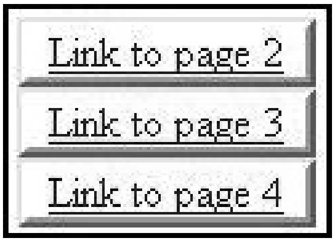


Figure 5.13: Centered links with outset borders of a specific width.

What Properties Control the Background of an Element?

Use the **background-image** property to indicate the URL of an image file you wish to use as the background of the element. When specifying the URL, be sure to code it within the **url**('…') wrapper, as shown in the syntax below.

Set the **background-attachment** property to **fixed** if you wish the background image to remain in exactly the same position within the viewable area of the Web page. If this property is set to **scroll**, the image will move with the content as the user scrolls through the element.

If you are not using a background image and would rather set the background to a solid color, use the **background-color** property. Set it to the desired color name, a hex code, or an RGB value.

Tags

```
{background-attachment: fixed
       scroll
 background-color: color
      rgb(r,g,b)
      #(rr,gg,bb)
 background-image:url('url')
 background-position:
   x y
   top left
   top center
   top right
   center left
   center center
   center right
   bottom left
   bottom center
   bottom right
 background-repeat: repeat
     repeat-x
     repeat-y
     no-repeat
 background: color
    image
    repeat
    attachment
    position
}
```

If you want a background image to repeat (tile) to fill the element, set the **background-repeat** property. The property's default is **repeat**, which causes the image to tile as many times as needed to fill the available space. If it is set to **repeat-x**, the image repeats across the element horizontally, but only one row is tiled. If the property is set to **repeat-y**, the image repeats vertically, but only one column is tiled. To cause the image to only appear once, use the **no-repeat** value.

The **background-position** property identifies the starting position of the background image. Specify the *x* and *y* distances, where *x* is the offset distance from the left border, and *y* is the offset from the top border. Use the standard size values discussed earlier in this chapter. To avoid hardcoding a specific distance, you can specify one of the special values, such as **top left** or **center center**. As you would expect, the image is anchored to the element at the specified location. To set all the background image properties at once, use the **background** property and provide the values in the order shown.

To update the background of our sample Web page, we might remove all the background properties set in the **<body>** tag in the HTML code, and add the code in Figure 5.14 to the cascading style sheet.

```
body {background:url(barn5.jpg);background-repeat:no-repeat;
    background-position:top right;
}
```

Figure 5.14: The style sheet code for the body of a Web page.

This style defines the background as a single image (barn5.jpg), positioned in the upper-right corner of the page. This doesn't change the look of the page at all, but it does move the control of the background image into the cascading style sheet and out of the HTML code. There is another advantage to this that we'll discuss at the end of this chapter.

What Properties Control the Appearance of Elements?

The **list-style-type** property sets the specific symbol to use for ordered and unordered lists. The **disc**, **circle**, and **square** values create the standard symbols discussed in chapter 2. Set the type to **lower-roman** to use lowercase Roman numerals in an ordered list, or **upper-roman** for uppercase. **Upper-alpha** creates an ordered list with uppercase alphabetic characters, and **lower-alpha** uses lowercase. **List-style-position** has two values: **inside** indents the list items, while **outside** (the default) prints them aligned to the current text.

Tags

```
{list-style-type:
   disc
   circle
   square
   decimal
   decimal-leading-zero
   lower-roman
   upper-roman
   lower-alpha
   upper-alpha
   lower-greek
   lower-latin
   upper-latin
   Hebrew
   armenian
   georgian
   cjk-ideographic
   hiragana
   katakana
   hiragana-iroha
   katakana-iroha
 list-style: type position image
 list-style-position: inside
        outside
 list-style-image: url(url)
 marker-offset: length
    auto
}
```

If the standard symbols are not sufficient for you, use the **list-style-image** property to specify a URL containing an image file of a symbol. Be sure to wrap the URL with **url(**...**)**. Many people report having difficulty in getting **list-style-image** to work correctly and consistently in multiple browsers. If you have this problem, first try setting the position to **outside**. If that does not fix the problem, you might need to consider attaching a background image to the text instead of using an **<li>** tag.

The **marker-offset** property exists in the CSS definition from the W3C, but few, if any, browsers support it at this time.

To set the style, position and/or image at once, use the **list-style** property, and then provide a type, position, and URL. Any of these may be omitted, as needed.

The page properties affect the way an HTML document is printed. The two most common properties are **page-break-before** and **page-break-after**. Add one or the other of these to an element to control how it handles page breaks.

Tags

```
{marks: crop
   cross
 orphans: number
 page-break-after: auto
      always
      avoid
      left
      right
 page-break-before: (as above)
 page-break-inside: auto
      avoid
size: length width
   auto
   landscape
   portrait
 widows: number
}
```

To force a page break before a given element prints, set its **page-break-before** property to **always**. This forces a page break no matter how far down the page it is. Set a **page-break-inside** property to prevent page breaks from occurring within a given element.

Use the **orphans** property to set the minimum number of lines that must print at the bottom of a page before advancing to the next page. Similarly, **widows** defines the minimum number of lines that may print on a new page after a page break occurs in the middle of an element.

Set the **marks** property to **crop** if you want to allow images to print all the way to the edge of the paper, ignoring margins. Set this property to **cross** if you want to print alignment crosses on the paper, which are special symbols used by certain printers to guarantee correct alignment.

You may define the basic layout of the page by setting the **size** property to either **landscape** or **portrait**. **Auto** uses the default page size. If you want to manually set the page size, simply provide the length and width sizes.

There are also specific CSS properties for working with tables. For example, **table-layout** can be set to **fixed** if you have consistent row and column sizes. The sizes of the cells in the first row of the table provide the template that all subsequent rows use. This can lead to a performance improvement when loading large tables, as the browser does not need to calculate the size of each cell as it is displayed.

Tags

```
{border-collapse: collapse
      separate
 border-spacing: length
      horz vert
 caption-side: bottom
     left
     right
     top
 empty-cells: show
   hide
 table-layout: auto
     fixed
}
```

Use the **empty-cells** property to control how empty table cells are handled. Set this to **hide** if you want empty cells to be hidden from view, or **show** (the default) to indicate that all cells should be visible.

The **border-collapse** property compresses two adjacent borders into a single border, creating a more compact table. Setting this value to **separate** (the default) displays the two adjacent borders with a small space between them. Use the **border-spacing** property to control the size of the space between the borders when they are separated. Set **border-spacing** to a single value, and that length will be used as the size for both the vertical and the horizontal borders. Alternatively, provide the horizontal and vertical border sizes separately. The **caption-side** property simply determines the side of the table on which the caption appears. This can be set to **top**, **right**, **left**, or **bottom**.

Defining Style Classes

So far, you have seen ways to change the properties of standard HTML elements. There is a far more powerful option within cascading style sheets, however. You can define something called a *class*, which is a set of properties that can be used by one or more types of elements. You can define a class as a subclass of a specific element, as shown in Figure 5.15.

```
P.question {font-weight:bold;
}
P.answer {font-weight:normal;
}
```

Figure 5.15: Examples of subclasses.

You can then write some HTML code that uses the style sheet, as in Figure 5.16. As you can see, the style sheet defines two types of paragraph tags: the question class and the answer class. Questions will be displayed in bold, while answers will be displayed in normal text.

```
<p class="question" >How do we define style classes?</p>
<p class="answer">Using the ".class-name" syntax in the style
  sheet</p>
```

Figure 5.16: HTML code controlled by subclasses.

The beauty of this method is that if you decide you want to display all questions in, say, blue, and the answers in green, you would only have to update the style sheet, and all pages that reference it would be updated. The HTML code references the class by adding the **class** property to a tag. If there is a paragraph class defined with that name, its properties will be used on this element.

You also have the option to define classes that are completely independent of all elements. The code for this is shown in Figure 5.17. As you can see, a class is defined with a period (.) as its first character.

```
.question {font-weight:bold;
}
.answer {font-weight:normal;
}
```

Figure 5.17: Examples of classes.

Since the question class in Figure 5.17 is not associated with a specific HTML tag, any tag could inherit its properties by simply referring to it. This allows multiple elements, such as paragraphs and headings, to acquire the same properties.

How Do Elements Inherit Properties from a Parent Element?

Elements inherit any properties they can from their parent. So, if the **<body>** tag has its **font-family** set to Arial, every element within the body that prints text will acquire that font property by default. However, any child element that defines its own **font-family** property supersedes the value in the **<body>** tag.

It's also possible to use classes in style sheets to define properties for child elements. For example, suppose you created a list of questions and answers using the definition-list HTML tags. You could use the style classes shown in Figure 5.18 to define the look of the definition terms and the definition descriptions. You could also provide a separate look for the major and minor questions.

```
.major {background:gray;font-size:120%; width=200px;
}
.minor {background:white; font-size 90%;
}
.question {color:blue;
}
.answer {color:black;
}
```

Figure 5.18: Styles for a question-and-answer page.

The styles in Figure 5.18 define the major topics as having a gray background and larger text, while the minor topics have a white background and smaller text. Each cell within the row will inherit these properties from its parent. They will also define their own class as being either a question, which is printed in blue, or an answer, which is printed in white. The code for an HTML table that uses these classes is given in Figure 5.19.

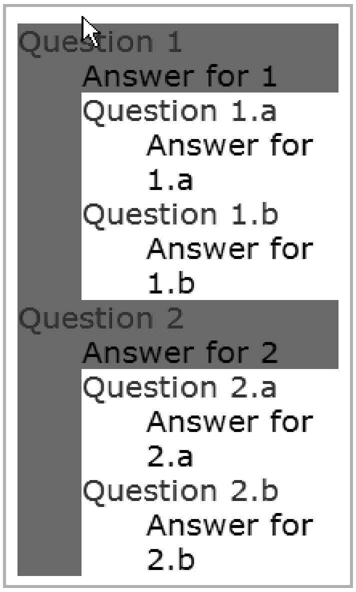
```
<dl class="major">
<dt class ="question">Question 1</dt>
<dd class ="answer">Answer for 1</dd>
<dl class="minor">
<dt class ="question">Question 1.a</dt>
<dd class ="answer">Answer for 1.a</dd>
<dt class ="question">Question 1.b</dt>
<dd class ="answer">Answer for 1.b</dd>
</dl>
<dt class="question">Question 2</dt>
<dd class ="answer">Answer for 2</dd>
<dl class="minor">
<dt class ="question">Question 2.a</dt>
<dd class ="answer">Answer for 2.a</dd>
<dt class ="question">Question 2.b</dt>
<dd class ="answer">Answer for 2.b</dd>
</dl>
</dl>
```

Figure 5.19: The HTML code for a question-and-answer page.

The **<dl>** tag defines the beginning of a new definition list. In this case, it also selects the class as being major or minor. The **<dt>** tag identifies a term, or in this case a question, and defines the class as being a question. The **<dd>**, or definition description, tag uses the answer class. The text that appears in the list acquires the properties of both the classes set in the **<dt>** and **<dd>** tags, as well as the class set in the **<dl>** tag. This is because the **<dt>** and **<dd>** tags are within the **<dl>**, making them child elements that inherit properties from their parent.

The code in Figure 5.19 creates the page shown in Figure 5.20. All of the questions appear in blue, while the answers are in black. The major questions appear in a larger text with a gray background. The **width** property in the major class limits the width of the element to 200 pixels. Without this, the gray background would extend all the way across the page.

You might be thinking that the list in Figure 5.20 is pretty ugly. While that's true, it provides a fairly simple way to illustrate the inheritance we're discussing, and that is our primary goal.



Question 1
Answer for 1
Question 1.a
Answer for 1.a
Question 1.b
Answer for 1.b
Question 2
Answer for 2
Question 2.a
Answer for 2.a
Question 2.b
Answer for 2.b

Figure 5.20: A question-and-answer HTML page, showing definitions with inheritance.

Rather than using inheritance to provide the flexibility needed in the question-and-answer list, we could have assigned multiple classes to a single element. The code for the cascading style sheet remains the same as in Figure 5.18, but the HTML changes to that in Figure 5.21.

```
<dl>
<dt class ="question major">Question 1</dt>
<dd class ="answer major">Answer for 1</dd>
<dl class ="minor">
<dt class ="question minor">Question 1.a</dt>
<dd class ="answer minor">Answer for 1.a</dd>
<dt class ="question minor">Question 1.b</dt>
<dd class ="answer minor">Answer  for 1.b</dd>
</dl>
<dt class ="question major">Question 2</dt>
<dd class ="answer major">Answer for 2</dd>
<dl class ="minor">
<dt class ="question minor">Question 2.a</dt>
<dd class ="answer minor">Answer for 2.a</dd>
<dt class ="question minor">Question 2.b</dt>
<dd class ="answer minor">Answer for 2.b</dd>
</dl>
```

Figure 5.21: Alternative HTML code for the question-and-answer page.

In this version of the definition list, all of the decisions about question versus answer and major versus minor have been moved into the specific **<dt>** and **<dd>** tags. To improve the look of the list, we also included the minor class for the inner (minor) definition lists. As shown here, more than one class may be listed for the **class** property of an HTML tag. Simply list all of the relevant classes, with a space between each. The result of this modified code, shown in Figure 5.22, is noticeably different than the previous example.

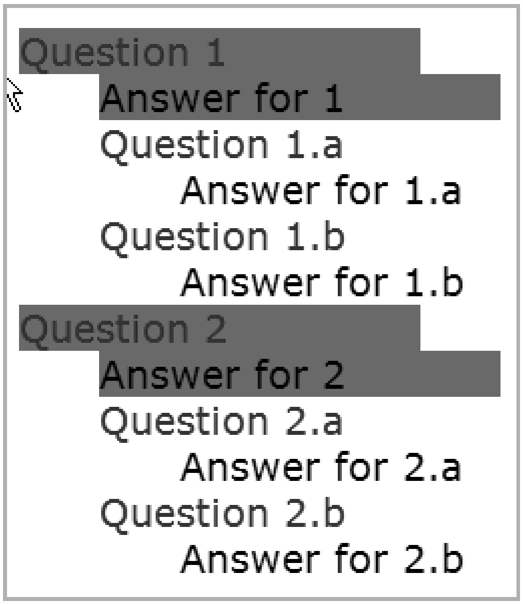


Figure 5.22: A definition list with multiple classes.

Since we moved the "major" class out of the **<dl>** tag and into the **<dt>** and **<dd>** tags, only those specific question-and-answer sections have a gray background. Since the answer is indented beneath the question, and both have a fixed length of 200 pixels, the gray shading for the answer boxes sticks out further than the questions. We might want to set a smaller width for the answer so that they align on the right side, but that is a cosmetic change that we don't need to worry about here. You can do it on your own, if you want.

It is also possible to define the style for elements that are the child of other specific elements. For example, you could define the look of the **<li>** tag when it is within an ordered list as being different from when it is within an unordered list. The CSS code might look like Figure 5.23.

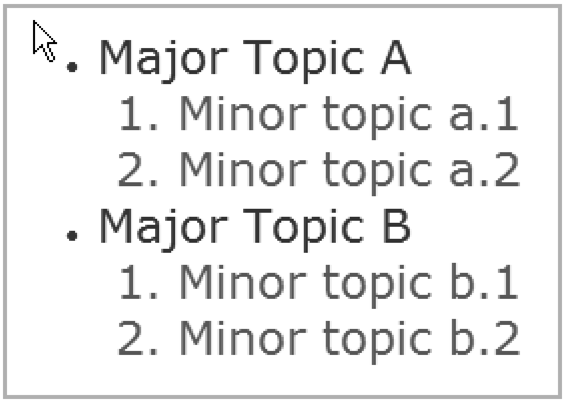
```
UL LI {color:blue; font-size:120%;
}
OL LI {color:green; font-size:100%;
}
```

Figure 5.23: Styles for list items within unordered and ordered lists.

Using this style sheet, the sample HTML code shown in Figure 5.24 creates the output in Figure 5.25.

```
<ul>
<li>Major Topic A</li>
  <ol>
  <li>Minor topic a.1</li>
  <li>Minor topic a.2</li>
  </ol>
<li>Major Topic B</li>
  <ol>
  <li>Minor topic b.1</li>
  <li>Minor topic b.2</li>
  </ol>
</ul>
```

Figure 5.24: The HTML code to create the Web page in Figure 5.25.



```
• Major Topic A
    1. Minor topic a.1
    2. Minor topic a.2
• Major Topic B
    1. Minor topic b.1
    2. Minor topic b.2
```

Figure 5.25: Nested lists with subclass styles.

Remember that in this example all six lines of code are generated within the **<li>** tag. The different behavior comes from the parent element, in this case, the **<ul>** or **<ol>** tag. This type of subclass is called a *child selector*.

Another type of a subclass is the *descendant selector*, which identifies an element that is descended from another element, but not necessarily an immediate descendant. For example if you added definition lists within the minor topics in the previous examples, you would have several layers of elements. You would have an **<ol>** containing a **<ul>** containing a **<dl>**. To assign a style to a definition list contained somewhere inside an ordered list, you would use the code shown in Figure 5.26.

```
OL>DL {color:green; font-size:100%;
}
```

Figure 5.26: The style for a definition list somewhere within an ordered list.

If you needed to define the style of a definition list that was the grandchild or later descendant of another element, you would use the code in Figure 5.27 in the CSS. In this example, if the definition list were immediately beneath the ordered list in the HTML code, this style would not apply. There would have to be at least one other element between them for this style to apply.

```
OL*DL {color:green; font-size:100%;
}
```

Figure 5.27: The style for a definition list that is at least a grandchild of an ordered list.

If you needed to identify sibling elements, such as a paragraph that immediately follows an **<h3>** tag, you could define a style sheet as shown in Figure 5.28. In this case, the plus sign indicates that the **<p>** tag must follow the **<h3>** tag. It is important to note that as opposed to the earlier examples, the **<p>** tag is not inside the **<h3>** tag, but adjacent to it, within a larger element such as the page body.

```
H3+P {color:blue; font-size:100%;
}
```

*Figure 5.28: The style for a paragraph that immediately follows an **<h3>** tag.*

Once you understand these basic methods of identifying various selectors based on their relationships, an even more complex method of identifying elements in relationships to one another is to string multiple dependent selectors together. For example, if you wanted to identify only those ordered lists that existed somewhere within a paragraph and immediately after an unordered list, you could write the CSS code shown in Figure 5.29.

```
P>UL+OL {color:blue; font-size:100%;
}
```

Figure 5.29: An example of nested subclasses.

This might seem at first like an odd and not terribly useful ability. However, what if you wanted to nest one unordered list inside another, such as shown in Figure 5.30?

```
<ul>
<li>Major Topic A</li>
  <ul>
  <li>Minor topic a.1</li>
  <li>Minor topic a.2</li>
    <ul>
    <li>Tertiary topic a.2.1</li>
    <li>Tertiary topic a.2.1</li>
   </ul>
  </ul>
</ul>
```

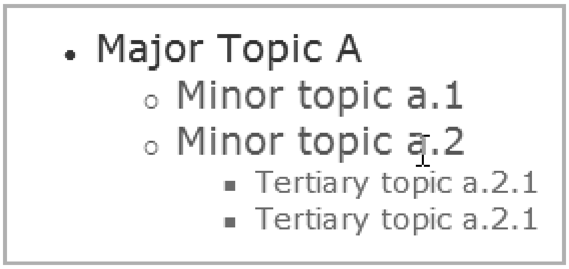
Figure 5.30: The HTML code for nested lists.

Normally, for nested lists like this, the browser will assign different symbols such as disk, circle, and square to the list items at each level. But what if you wanted to change more than the symbol? What if you wanted to change, say, the text color and size as well? You could use the CSS code in Figure 5.31 to define the behavior for each layer of nested, unordered lists.

```
ul {color:blue; font-size:120%;
}
ul ul {color:green; font-size:100%;
}
ul ul ul {color:Red; font-size:80%;
}
```

Figure 5.31: Styles for nested lists.

The first style is for the top-level unordered list. If you didn’t code anything else, it would apply to all levels. The second style applies to the second level of nested unordered lists. The third style applies to all unordered lists that have been nested at least three levels deep. When this CSS code is combined with the previous HTML code, it generates the output shown in Figure 5.32. To continue defining different looks for deeper levels such as the fourth or fifth levels, simply add additional styles with either four or five **ul** identifiers at the beginning.



- Major Topic A
    - Minor topic a.1
    - Minor topic a.2
        - Tertiary topic a.2.1
        - Tertiary topic a.2.1

Figure 5.32: Three nested, unordered lists.

Whenever more than one selector applies to an element, the one that is most precise takes precedence. There is a formula for determining this, but it's a bit more complicated than we want to try to explain here, so the overly simplified rule is this:

> The selector that refers to the most IDs (discussed next) takes precedence. If the selectors refer to the same number of IDs (or none), then the number of classes referenced determines the selector that takes precedence. If the selectors refer to the same number of classes (or none), then the number of HTML tags referenced determines the selector that takes precedence. If two selectors refer to the same number of HTML tags, then the one listed last in the style sheet takes precedence.

What Is an ID?

So far, we have been talking almost exclusively about classes. But there is another entity called an *ID*. Where classes are used to define styles for one or more HTML elements, IDs are exclusively designed to uniquely identify a single element. So, if a page had three paragraphs, you might assign the same class to all three, but also assign a unique ID to each one, as shown in Figure 5.33.

```
<p id="p1" class="notes">Some misc information</p>
<p id="p2" class="notes">More misc information</p>
<p id="p3" class="notes">The last misc information</p>
```

Figure 5.33: Paragraph tags with IDs.

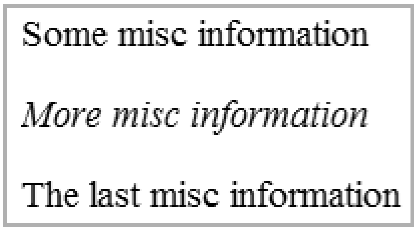
Any attributes that are common to all three paragraphs can be assigned via the class's style. If you wanted only the second paragraph to print in italics, however, you might create CSS code such as that shown in Figure 5.34. The **.notes** entry defines the style for the notes class, which then applies to all three paragraphs. All ID tags referenced in a CSS precede the **id** identifier with a pound sign (#). So, the entry starting with **#p2** defines the style for the second paragraph. Because the HTML code has both a class and an ID, both styles apply. The entry for the **id** tag overrides any conflicting values from the notes class.

```
.notes {font-style:normal; width:60%;margin-left:20%;
}

#p2 {font-style:italic
}
```

Figure 5.34: The style for the notes class and the p2 ID.

The three paragraphs in Figure 5.33, with the CSS code in Figure 5.34, create the output in Figure 5.35. Remember that when you use an **id** tag, it should be unique within the HTML document.


Some misc information

More misc information

The last misc information

Figure 5.35: Three paragraphs with classes and IDs.

What You Can Do with a Cascading Style Sheet

You can define style rules for virtually every HTML element, even if some are pointless, such as a font assignment for an embedded video file. All of the examples you've seen so far are fairly simple. This is appropriate, because you are just trying to figure out how these things work.

One of the main uses for style sheets is to control the arrangement of content on the page. By *arrangement*, we mean not just font sizes, colors, etc., but to actually move elements all over the page. One of the neat things you can do is create pages that can wildly change their looks and layouts, simply by changing the cascading style sheet they use. We'll discuss this further in chapter 6. For now, let's look at a revised version of our Web page that incorporates a cascading style sheet. We'll change the page to pull in the barn image as a background.

The modified HTML code is shown in Figure 5.36. Nearly all the line breaks, the width attributes, and even the background image have been removed. The previous background image was the size of the entire page, with white space to the left and bottom. By moving the background image's definition to a cascading style sheet, we have better tools for controlling its behavior, and we can eliminate the white space and use a smaller image that includes just the barn. Because it is smaller, the image will load faster than on the previous page. You will also notice the class definitions sprinkled throughout the code.

```
<html>
<head>
<link href="bbqtheme.css" rel="stylesheet" type="text/css">
</head>
<body>
<h1 class="heading1" >Bill's Barbeque Barn</h1>
<H3><P class="intro">Here at Bill's BBQ Barn, there is nothing
  we like more than sharing our Blue Ribbon BBQ Recipe with all
  our friends.
</P>
<P class ="intro">
Join us for some of the best BBQ you'll find anywhere. After you
  sample some of our famous food, stop by the gift shop and pick up
  a bottle of Bill's BBQ Sauce to take home, or find the perfect
  gift for your Backyard BBQ Grill Master at home.
</P>
<P class ="intro">
We think that you'll have to agree with us, that there is nothing that
brings back that "down home" feeling like a great BBQ dinner. And
  Great BBQ is what we do.
```

Figure 5.36: The modified HTML code for the BBQ Barn home page (part 1 of 2).

```
</P></H3>
<BR><BR><BR>
<HR>
Learn More about:<br>
<aclass="inset"href="BarbequeBarna.html"target="_blank">Country
  Store</a>
<a class="inset" href="BarbequeBarnb.html" target="_blank">BBQ
  Flavors</a>
<aclass="inset"href="BarbequeBarnc.html"target="_blank">"How
  To" BBQ</a>
</body>
</html>
```

Figure 5.36: The modified HTML code for the BBQ Barn home page (part 2 of 2).

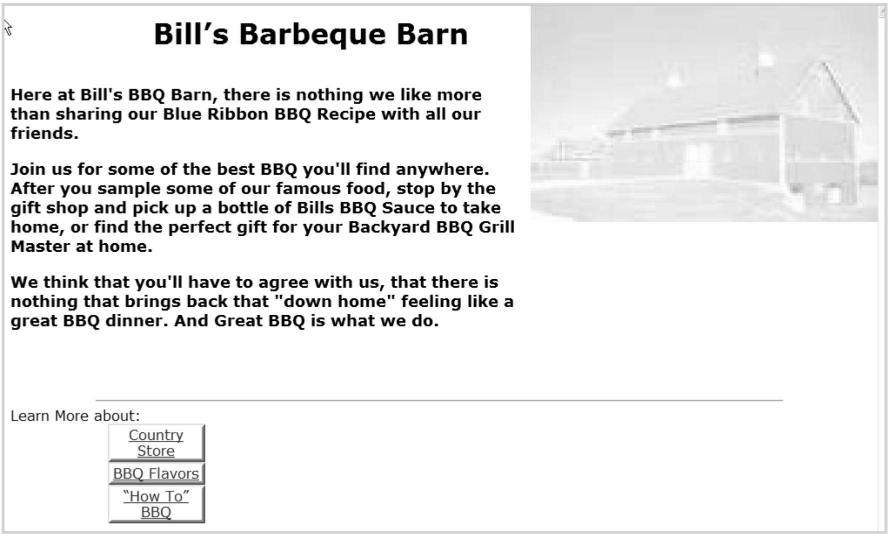
The new cascading style sheet for our home page is shown in Figure 5.37. This cascading style sheet defines the picture of the barn as the background for the page, positions it in the upper-right corner, and prevents it from repeating, so only one barn is shown.

```
body {background:url(barn5.jpg);background-repeat:no-repeat;
   background-position:top right;
}
.heading1 {width:70%; text-align:center; margin-bottom:40px;
}
.intro {width:60%;
}
hr {width=80%; text-align:center;height:2px;
}
.inset {margin-left:7em; width:80%;
}
a.block {display:block; border-style:outset;
    width:14ex;text-align:center; margin-left:7em;
}
```

Figure 5.37: The cascading style sheet for the BBQ Barn home page.

We've added a bottom margin to the first heading, so the text that follows appears farther down the page. The introductory section has its width defined as 60% of the page, so it won't overlap the background image. Virtually all of the properties of the horizontal rule, including its height, width, and alignment, are controlled in the cascading style sheet.

The **block** class defines the style for the hypertext links at the bottom of the page. It defines an indention, a width, a border style, and an alignment. As noted earlier in the chapter, the **display:block** property forces a new line after each link. The **outset** border makes the links appear like buttons on the page. The resulting Web page is shown in Figure 5.38.



Bill’s Barbeque Barn

Here at Bill's BBQ Barn, there is nothing we like more than sharing our Blue Ribbon BBQ Recipe with all our friends.

Join us for some of the best BBQ you'll find anywhere. After you sample some of our famous food, stop by the gift shop and pick up a bottle of Bills BBQ Sauce to take home, or find the perfect gift for your Backyard BBQ Grill Master at home.

We think that you'll have to agree with us, that there is nothing that brings back that "down home" feeling like a great BBQ dinner. And Great BBQ is what we do.

Learn More about:

Figure 5.38: The updated Web page using a cascading style sheet.

Summary

Cascading style sheets provide a tremendous amount of flexibility and complexity to Web page design. Combining HTML with style sheets creates vastly more sophisticated Web pages, and the whole thing begins to feel more like programming.

Most programmers will, or at least should, partner with experienced Web designers to create the layout and look of their Web pages. If you don’t have an experienced designer to learn from, there are countless online tutorials and books on cascading style sheets. With a little research and effort, you can build on this introduction to cascading style sheets, and you’ll be on the road to becoming an experienced Web developer yourself!