CHAPTER 26

Learning CSS Syntax

In This Chapter

If you look at a style sheet and think you'll never get the hang of it, remember how you felt when you looked at HTML for the first time. Now you have no fear of it because you understand the rules; you know that even if you see an element you've never seen before, you can figure it out. CSS isn't any more difficult. It has a clear set of rules that, unlike the set of rules for HTML, fits into one chapter. To help you along, a complete list of all the properties you can use in CSS is in Appendix E.

By the time you finish this chapter, you'll know the syntax of CSS, how CSS differs from HTML, and how to define and group CSS properties. You learn some shortcuts so your property definitions won't be quite as verbose as the example in the previous chapter. You also learn about the box-formatting model CSS uses and about inheritance, which is why you don't need to redefine every property for every element. You also learn about defining classes and IDs, which enable you to add variety to different instances of the same element on the same page. Finally, you see the `DIV` and `SPAN` elements in action.

Anatomy of a Style Sheet

CSS has its own vocabulary. To understand what goes where in a style sheet, you need to understand what each item on the style sheet is called. A style sheet is composed of *rules*. This is a rule:

```
P {
        text-indent: 2cm;
        color: black;
        padding-top: .25in;
}
```

In the previous example, `P` is the *selector*. The selector could just as easily have been P.special, or P#123. A selector indicates to which elements, which class of elements, or which IDs of elements the rule applies.

A *rule* is made up of a selector with one or more declarations. Each of the three lines under the selector is a declaration. A *declaration* is composed of a property and one or more values. The *property* is separated from the value by a colon. Each declaration ends with a semicolon. All the declarations are listed within curly braces ({}).

Differences Between CSS and HTML Syntax

When learning HTML, you learned HTML has *elements* and that *elements have attributes*. CSS has selectors, which you should recognize, because they can be the same as the elements you use in HTML. In CSS, however, instead of the selectors having attributes, they have properties. The properties of CSS selectors, just like the attributes of HTML elements, have values.

Here is an example of an HTML element with one attribute:

```
<BODY dir="ltr">
This is my very short page.
</BODY>
```

and this is an example of a CSS selector with one declaration (property-value pair):

```
BODY {
       color: white;
}
```

You can see that, just like an HTML element definition, the CSS selector has the element name in it. This is where the similarities end, though. In CSS, there are no angle brackets (<>). Instead, a CSS rule has curly braces around the declaration or declarations. The CSS selector name is not enclosed with any kind of markings.

There are no start or end tags, and no content. The reason for this should be obvious: CSS documents don't have anything rendered. Instead, they just have information *about* what is rendered in the HTML document. CSS documents have properties that describe how different parts of the element will be rendered. These properties have values, just like attributes in HTML have values. The attribute-value pair in HTML looks like this:

```
colspan="3"
```

In CSS, a property-value pair, which is called a *declaration*, looks like this:

```
color: white;
```

The three differences between defining attribute-value pairs and defining property-value pairs are as follows:

1. Property-value pairs use a colon instead of an equal sign.
2. In CSS, double quotation marks do not enclose the value (unless the value is multiple words, as in a font name).
3. A semicolon follows the property-value pair.

Once you get used to those differences, you'll write declarations in CSS as deftly as you write attributes in HTML.

Defining Properties

CSS is flexible. You can define a selector with one declaration or a selector with many declarations. You use the same syntax either way. The basic outline of a CSS rule is as follows:

```
SELECTOR-NAME {
         property: value;    /* declaration */
         property: value;
         property: value;
}
```

If a value has multiple words, the value needs to be enclosed in double quotation marks.

Here is an example of a style sheet with two rules:

```
BODY {
       font-family: "Times New Roman",
  "Times Roman",
  serif;
       color: black;
       background: white;
       margin-left: .5in;
       margin-right: .5in;
}
P {
       text-indent: .5in;
       margin-top: .25in;
}
```

All it does is set the body of the document to have a white background with left and right margins of a half inch. It also sets the text to be black. The font-family property tells the browser to use the first font in the list that it finds on the client computer. Notice multiword font names, such as Book Antiqua and Times New Roman, each

have double quotation marks around them. The last name in the list is a font type, rather than a specific font. If neither of the first two fonts are on the client computer, it says, use any serif font. This style sheet also formats paragraphs to have an indentation on the first line of paragraphs of a half inch, and a top margin of a quarter inch. This means a quarter inch of white space will be between paragraphs.

Grouping Properties

Here's another style sheet that defines the font family for headings:

```
H1 {
        font-family:  Helvetica,
                      Arial,
                      sans-serif;
}
H2 {
        font-family:  Helvetica,
                      Arial,
                      sans-serif;
}
H3 {
        font-family:  Helvetica,
                      Arial,
                      sans-serif;
}
```

What it says is headings `H1`, `H2`, and `H3` should use Helvetica, if it is available; Arial, if Helvetica is not available; and any sans-serif font, if neither Helvetica nor Arial are available. To say all this takes a lot of space, though.

Fortunately, a more concise way exists. You can group property definitions by placing the element names into a comma-delimited list as follows:

```
H1, H2, H3 {
        font-family:  Helvetica,
                      Arial,
                      sans-serif;
}
```

Using this technique of grouping elements is a good way to save space, increase readability, and reduce the possibility of introducing errors.

Property Definition Shortcuts

You can define a lot of properties for fonts, as this style-sheet excerpt demonstrates:

```
H3 {
        font-weight: bold;
        font-size: 16pt;
        line-height: 20pt;
        font-family: "Times New Roman";
        font-variant: normal;
        font-style: italic;
}
```

Fortunately, you can string this all into one property, called *font*, which shortens the definition considerably.

```
H3 {
        font: bold 16pt/20pt "Times New Roman" normal italic;
}
```

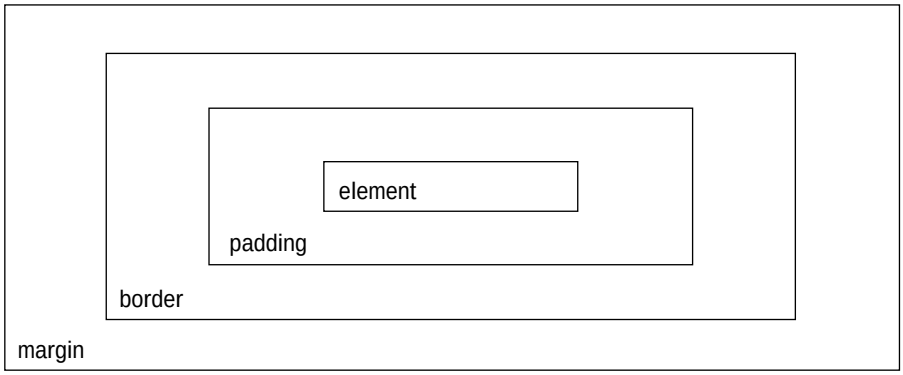
Once you get used to the shortcut, you'll never use the verbose definition again.

Box Formatting: The CSS Formatting Model

CSS uses a clever metaphor for helping you specify containers (block-level elements) on your page: the box. When you define formatting for your block-level elements — whether they be paragraphs, blockquotes, lists, images, or whatever — for purposes of CSS, you are defining formatting for a box. It doesn't care what is in the box; it just wants to format the box.

Box dimensions

The first thing the browser does is render the block-level element to determine what the physical dimensions of the element are, given the font selected for the element, the contents of the element, and any other internal formatting instructions supplied by the style sheet. Then the browser looks at the padding, the border, and the margins of the element to determine the space it actually requires on the page.



Padding is the distance between the outside edges of the element and the border. The *border* is a line or ridge. The *margin* is the distance between the border and the outer box of the next container. How you define the padding, border, and margin is described in detail in the following sections.

Padding

You don't need to define any padding, but if you are going to define a border, then you probably want to define padding so your element doesn't look too crowded. The default for an element is no padding. Figure 26-1 shows the same table with and without padding. You can see the one without padding looks crowded.

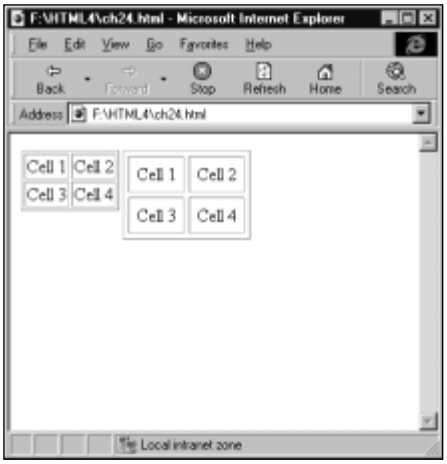


Figure 26-1: Tables with and without padding

Five properties are associated with padding. They are as follows:

1. `padding`, which gives the same padding on all sides
2. `padding-top`
3. `padding-right`
4. `padding-bottom`
5. `padding-left`

Get used to seeing the -top, -right, -bottom, and -left additions to property names. This is how all box-related properties are specified.

Suppose you want to define your paragraphs to have padding on the top, the left, and the right; you could use the following style sheet:

```
P {
        padding-top: .5in;
        padding-right: .5in;
        padding-left: .5in;
}
```

Or you could use shorthand to write out the padding properties as follows:

```
P {
        padding: .5in .5in 0in .5in;
}
```

You can always string the top, right, bottom, and left properties together in that order. The same shorthand works for margins and borders. Notice no commas are between the items in the list.

Border

The default is to have no border on elements. You can define a border in two different ways. Either you can define the width, color and style of the border, by side, or you can define the width, color, and style for the box individually. Two examples follow:

```
BLOCKQUOTE {
        border-width: 1pt 1pt 0pt 1pt;
        border-color: black;
        border-style: solid;
}
```

The previous example creates a black, solid border for the top, right, and left sides of the list.

```
BLOCKQUOTE {
        border-top: 1pt solid black;
        border-right: 1pt solid black;
        border-left: 1pt solid black;
}
```

Both these examples create the same border. The border is inserted between the padding, if there is any, and the margin, if there is any. Valid values for border style are: none, dotted, dashed, solid, double, groove, ridge, inset, and outset.

Or, if you want to create a border that is the same on all four sides, you can use the border property:

```
BLOCKQUOTE {
        border: 1pt solid black;
}
```

Margins

Margins create white space outside of the border. Notice in Figure 26-1 that the two tables are immediately adjacent to each other. This is because neither one has margins. Margins are created with the margin, margin-top, margin-right, margin-bottom, and margin-left properties. They work exactly the same as the padding property.

Understanding Inheritance

If you define a background color for the `BODY` of your document, you don't have to define the same background color for each `P` element. Why not? Inheritance. Elements within containers inherit characteristics of the containers in which they exist. A paragraph is a container, but it is also within a container — the `BODY` element. If you have an italicized word in your paragraph, you can expect that word to inherit the font, text color, text size, line spacing, and so forth of the paragraph (except it will be italicized); you can also expect the paragraph to inherit a few things from the `BODY`, such as background color, padding, margins, and anything else the `P` element doesn't specifically specify.

Inheritance is great. *Inheritance* enables you to define formatting only at the highest level. The formatting you define trickles down to the lower-level elements. An awareness of inheritance can keep you from specifying every property at every level.

Defining Classes

Classes are how you customize elements in your page. With classes, you can define more than one look for an element in your style sheet and then, in your page, you say which look you want to use. Classes are defined in your style sheet using the following notation:

```
P.first {
       font: bold 12pt/14pt "Times New Roman";
}
P.second {
       font: normal 12pt/12pt "Times New Roman";
}
```

To use these classes in your page, refer to them as follows:

```
<P class="first">This is a first-class paragraph.</P>
<P class="second">This is a second-class paragraph.</P>
```

Note the first class of `P` inherits all the formatting of `P` that it doesn't specifically override. The second class of `P` also inherits all the formatting of `P` that it doesn't specifically override. Classes first and second are not related to each other except they are both classes of the `P` element. You can make up your own names for classes.

Pseudo-Classes

When you are looking at a page and have already clicked one or more of the links on the page, you might notice the links you have already clicked are a different color from the links you have not yet clicked. You can define what you want those colors to be using some of the available pseudo-classes in style sheets. Even though *pseudo* means *fake* in Greek, these aren't really fake classes — they're predefined classes that already mean something to the browser. Pseudo-classes enable formatting based on characteristics other than name, attributes, or content. You can define pseudo-classes for three different types of links:

1. **A:link** for unvisited links
2. **A:visited** for visited links
3. **A:active** for the active link (that is, the link you are currently clicking)

If you want to add a regular class to the `A` element, you define the selector as `A.first-class:link`. Consider the following example:

```
A:link {
       color: red;
```

```
}
A:visited {
        color: blue;
}
A:active {
        color: green;
}
A.special:link {
        color: #FF33FF;  /* fuchsia */
}
```

This style sheet has four rules. The first three define the colors of unvisited links, visited links, and active links. The last one will only be used when the class is specified as special; it specifies the color of unvisited links.

Defining IDs

You won't use IDs nearly as often as you use classes, but it's nice to know they are there if you need them. *IDs* are like classes, except they are not necessarily associated with elements. Isn't this contrary to the HTML 4 Way? Yes, but it's there if you need it. IDs are defined as follows:

```
#wide {
        letter-spacing: .4em;
}
```

And used as follows:

```
<H1 id="wide">This is a wide heading</H1>
<P id="wide">This is a paragraph of widely spaced text.</P>
```

As you can see, the ID wide can be used with any element. It is recommended you use classes rather than IDs.

Grouping Elements with DIV and SPAN

In Chapters 16 and 17, you learned about using the `DIV` and `SPAN` elements to group block-level elements and inline elements, respectively. Now you can finally see how this works with style sheets. Consider the following style-sheet rules:

```
DIV.important {
        background: red;
        font: bold 14pt/18pt Helvetica;
}
```

```
SPAN.incidental {
       font: normal 8pt/8pt Helvetica;
background: gray;
}
```

You can use them as follows:

```
<DIV class="important"><P>This is one very important
paragraph.</P>
<TABLE>
...table contents...
</TABLE>
<P>This is another important paragraph.</P></DIV>
<P>This is somewhat important. <SPAN class="incidental">And
this is really incidental.</SPAN> But you might want to
remember this fact.</P>
```

This creates two paragraphs with a table between them that have a background color of red and 14pt bold Helvetica text. Following this mess is another paragraph with formatting inherited from the `BODY` element, but the middle sentence has gray 8 pt Helvetica text.

Comments in Style Sheets

Adding comments to your style sheets, just as you add comments to your HTML pages, is not a bad idea. Comments are defined in your style sheet using a different convention than in your HTML pages. Look back to the sample code for the pseudo-classes style sheet. Notice next to the hexadecimal value #FF33FF, there is the following:

```
/* fuchsia */
```

That is a comment in CSS. Adding a comment is little trouble when you create the style sheet; it is a lot more work to go back and remember what you meant when you defined the style sheet. If you use comments nowhere else, use them when you use hexadecimal notation for colors.

Comments are created with a /* preceding them and a */ following them.

From Here

Jump to Chapter 33 to learn about CSS positioning options

Proceed to Chapter 27 and begin adding styles to your Web page.

Summary

In this chapter you learned about the basics of CSS. You learned the vocabulary, the conventions, and differences between CSS and HTML syntax. You learned about defining properties, grouping properties, and defining properties using shortcuts. You learned about the CSS box model and about inheritance from containers to elements. You also learned about classes, pseudo-classes, and IDs. Finally, you learned about using the `DIV` and `SPAN` elements to group elements so they can all take advantage of the same formatting.

✦ ✦ ✦