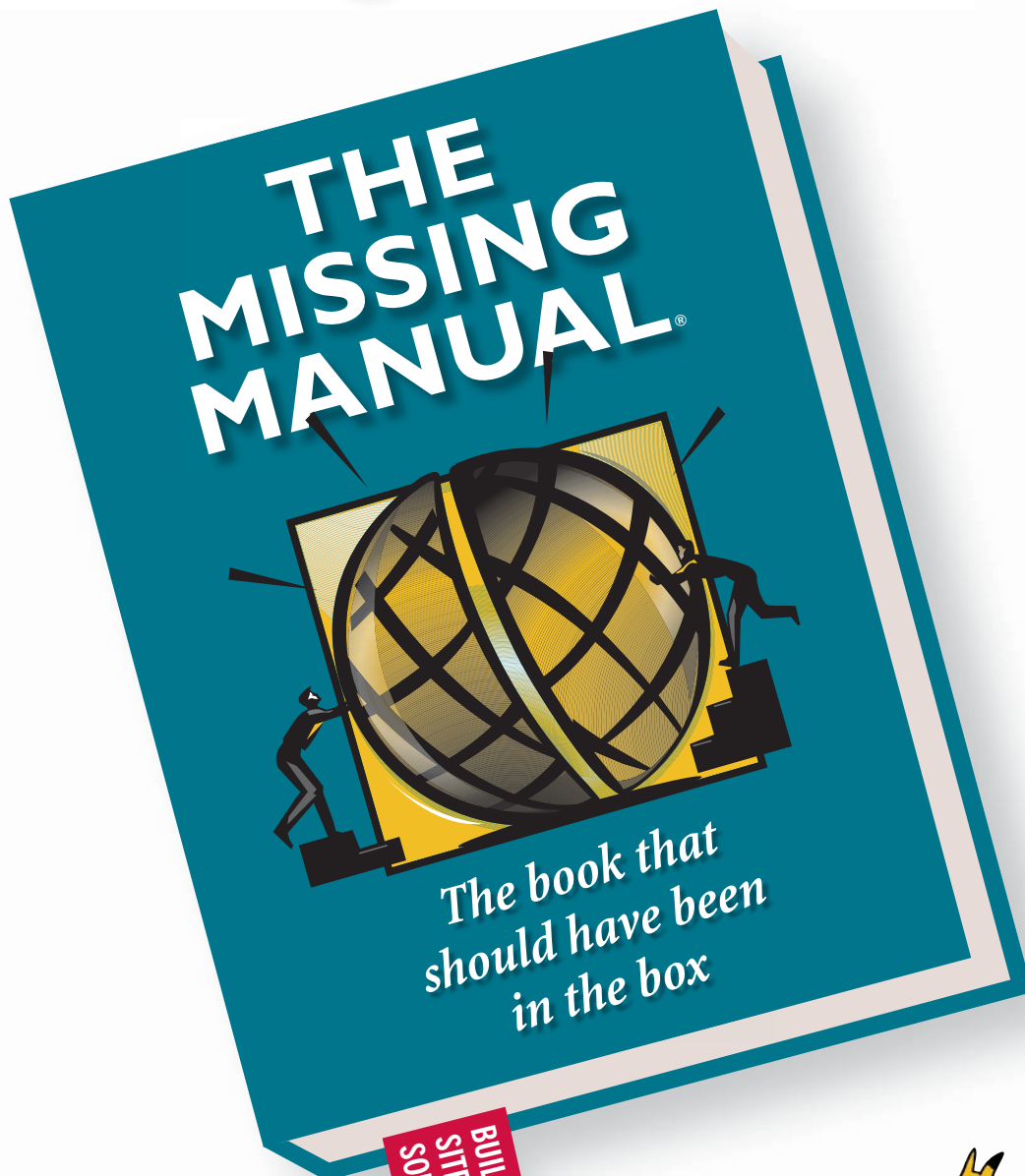


# Creating Web Sites

Covers  
HTML, CSS  
& BLOGS



Matthew MacDonald



POGUE PRESS™  
O'REILLY®

# Style Sheets

Last chapter, you learned HTML's dirty little secret—it doesn't have much formatting muscle. If you want your Web pages to look sharp, you need to add style sheets into the mix.

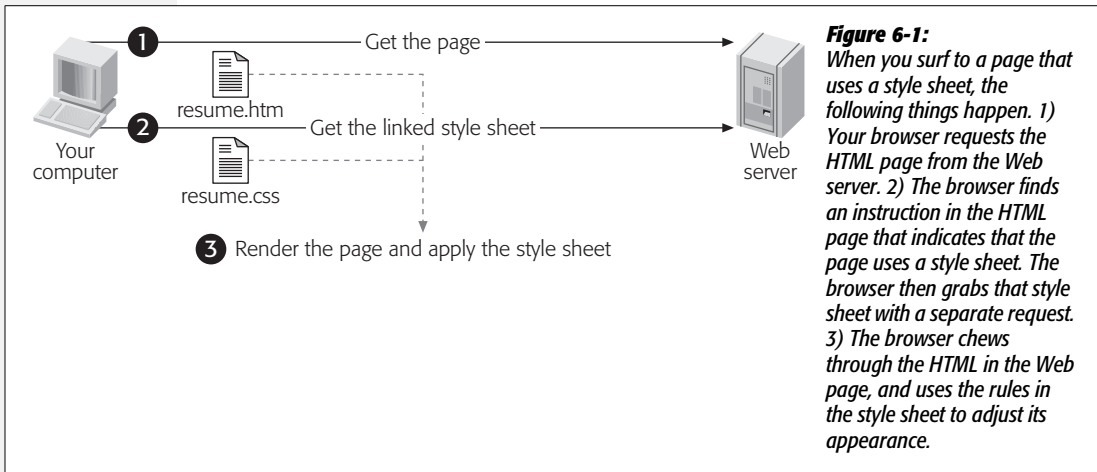
Style sheets are separate documents that are filled with formatting rules. The browser reads these rules and uses them to format your Web page. For example, a style sheet rule might say, "make all headings bold and fuchsia and draw a box around each one."

There's several reasons that you place formatting instructions in a style sheet instead of directly in a Web page. The most obvious one is *reuse*. For example, thanks to style sheets, you can create a single rule that you can use with every level three heading in every Web page on your Web site. The second reason is that style sheets help you make tidy, manageable HTML. Because they do all the formatting, your HTML code doesn't need to. All your HTML needs to do is organize the page into logical sections. (For a quick recap of the difference between structuring and formatting a Web page, refer back to page 112.)

The formatting choices in style sheets are much more extensive (and much more overwhelming) than those in HTML alone. Using style sheets, you can control colors, borders, margins, alignment, and (to a limited degree) fonts. You'll use style sheets in this chapter and throughout this book. As you'll see, style sheets give you options that can jazz up the dullest HTML.

## Style Sheet Basics

Style sheets are officially known as the *cascading style sheet* (CSS) standard. CSS is a system for defining rules about how one or more Web pages should be formatted. When you use CSS in a Web page, the browser reads both the page's HTML and the style sheet rules. It then uses the style sheet rules to format the page. Figure 6-1 diagrams the process.



This system gives Web weavers the best of both worlds—a rich way to format pages and a way to avoid mucking up your HTML beyond recognition. In an ideal world, the HTML document describes only the structure of a Web page (what's a header, what's a paragraph, what's a list, and so on), and the style sheet explains how to give that Web page a hot new look.

### The Three Types of Styles

Before you even get started learning how to write CSS rules, you first have to think about *where* you're going to place those instructions. CSS gives you three different ways to apply style sheets to a Web page:

- An **external style sheet** is a style sheet that's stored in a separate file. This is the most powerful approach, because it completely separates the formatting rules from your HTML pages. It also gives you an easy way to apply the same rules to many pages.
- An **internal style sheet** is a style sheet that's embedded inside an HTML document (it goes right inside the <head> element). You still have the benefit of separating the style information from the HTML, and if you really want, you can copy the embedded style sheet from one page to another (although it gets difficult to keep all of those copies synchronized). Really, the only time you'll use an internal style sheet is if you want to give someone a Web page in a single file—for example, if you're emailing someone a Web page.

- An **inline style** is a method for inserting style sheet language directly inside an HTML tag. You’ve already learned that it’s a bad idea to embed formatting inside a Web page document, because these details are ugly and long. However, you might occasionally use this approach to apply one-time formatting in a hurry. It’s not all that clean or structured, but it does work.

## UP TO SPEED

## The “Other Way” to Format a Web Page

Style sheets aren’t the only way to format a Web page—they’re just the most capable tool. But you’ve also got a few formatting options built right into the HTML tags you learned about in Chapter 5. For example, you can change a page’s background color or center text without touching a style sheet. For the most part, this book doesn’t use these formatting options, for several good reasons:

- **They’re patchy and incomplete.** Many features (like paragraph indenting and borders) are completely missing—no HTML tags exist to achieve these effects. Even worse, the model isn’t consistent—for example, you might be able to line up text in one type of tag, but not the text that’s contained in another type of tag. This makes the model difficult to learn and remember.
- **According to the HTML standard, these formatting options are deprecated.** That means that even though these formatting tweaks are still supported by most browsers, they’re considered obsolete by the official rulemakers of the HTML standard—that’d be the good people who work at World Wide Web Consortium (W3C). Many people didn’t like these fancy flourishes in the first place, but they were wedged in by over-eager software companies such as Microsoft and Netscape. Newer devices (for example, browsers on tiny mobile phones) are

more likely to ignore these instructions altogether. Even worse, if you use them, your hard-core Web designer friends won’t sit with you at restaurants.

- **They don’t allow you to easily reuse formatting changes.** So after you format one page, you need to start all over again to fix the next page. And so on, and so on, and so on.
- **They won’t work in XHTML.** Right now you might not be concerned about creating XHTML pages (page 47), but by using style sheets, you’ll simplify your life if you ever decide to switch your Web site over to this new Web standard.
- **Why learn something you don’t need?** Seeing as style sheets offer so much more power and flexibility, and now that style sheets are supported (with certain limitations) on virtually every browser around (old and new), it doesn’t make sense to waste time with something you’ll outgrow anyway.

CSS does have one strike against it. If you plan to support Paleolithic browsers like Netscape 4, you’ll run into quirks with many parts of the CSS standard. In that case, you’d be better off sticking with pure HTML.

These three levels give you the flexibility to either follow the CSS philosophy wholeheartedly (with external style sheets), or to use the occasional compromise (with internal style sheets or inline styles). Because the style sheet language is always the same, even if you use the “lazier” approach, like internal style sheets, you can always cut-and-paste your way to an external style sheet when you’re ready to get more structured.

## Browser Support for CSS

Before you embrace style sheets, you need to make sure that they work on all the browsers your visitors are using. But determining what browsers support cascading style sheets isn't as easy a question as it should be. The first problem is that there is actually more than one version of the CSS standard—there's both CSS1 and CSS2. This book concentrates exclusively on CSS1, because support for CSS2 is lacking in most browsers. (Internet Explorer is a notable laggard in this department.) But the real problem is that browsers don't necessarily support the entire CSS standard, and when they do, they don't always support it in exactly the same way. The discrepancies range from minor to troubling. As a result, in this book you'll focus on CSS properties that are known to be well supported among all the major browsers. (That said, don't forget to test your pages in a wide variety of browsers to be sure they're appearing correctly.)

As a basic rule of thumb, you can count on good all-around CSS support in Netscape Navigator 6, Internet Explorer 5 for Windows, Internet Explorer 4.5 for Macintosh, Opera 3.6, and any version of Firefox or Safari. In later browser versions, the support just gets stronger.

Many people who've used the Web for a few years still remember an earlier generation of browsers—namely, Netscape 4.x and Internet Explorer 4.x. Both of these browsers are unreliable when it comes to some of the fancier features in CSS. However, you're unlikely to run into these browsers anymore outside of a museum. If you're in doubt, just take a look at some recent browser statistics to see who's online.

Table 6-1 shows the log results—a record of which visitors are using which browsers over a period of several months—at a popular HTML teaching site, W3Schools. Given the fact that W3Schools doesn't cater to power users or computer professionals, this table's a pretty good yardstick for the kinds of browsers your visitors are likely to be using. Every browser in this table supports CSS.

**Table 6-1.** Current browser usage

2005	IE 6	IE 5	Opera 7/8	Firefox	Mozilla	Opera 8	Netscape 7
<b>July</b>	67.0%	6.7%	0.4%	19.7%	2.6%	0.8%	0.5%
<b>June</b>	65.0%	6.8%	0.5%	20.7%	2.9%	0.7%	0.6%
<b>May</b>	64.8%	6.8%	0.6%	21.0%	3.1%	0.7%	0.7%
<b>April</b>	63.5%	7.9%	1.0%	20.9%	3.1%	0.4%	0.9%
<b>March</b>	63.6%	8.9%	1.6%	18.9%	3.3%	0.3%	1.0%

---

**Tip:** For more up-to-date information, surf to [www.w3schools.com/browsers/browsers\\_stats.asp](http://www.w3schools.com/browsers/browsers_stats.asp).

---

Of course, statistics can be misleading. Web sites can attract wildly divergent groups of people, and you know your audience best. If you're still concerned about whether

a specific CSS feature is supported in a specific browser version, see the sidebar “A Browser Compatibility Reference.” Also, look for the Oldest Supported Browsers column in the property tables shown throughout this chapter. For example, if the Oldest Supported Browsers column indicates Internet Explorer 5, you can bet that the property won’t work (or works erratically) in Internet Explorer 4.5.

#### FREQUENTLY ASKED QUESTION

### A Browser Compatibility Reference

*How can I tell if a CSS feature is supported in a particular browser?*

If you’re a hard-core Web maven, you may be interested in one of the Web browser compatibility charts for CSS that are on the Web. Two good resources are [www.corecss.com/properties/full-chart.php](http://www.corecss.com/properties/full-chart.php) and [www.quirksmode.org](http://www.quirksmode.org). These charts specify which CSS features are supported in different browser versions.

But chart-reader beware: These tables also include many rarely used or new and poorly supported features (such as CSS2 features). For example, you probably don’t care that virtually no browser supports the *pitch range* property, which is used in conjunction with text-reading devices. Unfortunately, the CSS charts can cause panic in those who don’t know the standards. However, they can be handy if you need to check out support for a potentially risky feature.

## The Anatomy of a Rule

Style sheets contain one thing: *rules*. Each rule is a formatting instruction that applies to a part of your Web page. A style sheet can contain a single rule, or it can hold dozens (or even hundreds) of them.

Here’s a simple rule that tells the browser to display all <h1> headings in blue:

```
h1 { color: blue }
```

CSS rules don’t look like anything you’ve seen in HTML. But you’ll have no trouble with them once you realize all rules are built out of three ingredients: *selectors*, *properties*, and *values*. Here’s the format that every rule follows:

```
selector { property: value }
```

And here’s what each part means:

- The *selector* identifies what you want to format. The browser hunts down all the parts of the Web page that match the selector. For now, you’ll concentrate on selectors that match every occurrence of a specific tag. But as you’ll learn later in this chapter (page 173), you can create more sophisticated selectors that pick out just specific sections of your page.
- The *property* identifies the type of formatting you’re applying. Here’s where you choose whether you want to change colors, fonts, alignment, or something else.
- The *value* sets the property. This is where you bring it all home. For example, if your property is color, the value could be light blue or dead-salmon pink.

Of course, it's rarely enough to format just one property. Usually, you'll want to change several characteristics at the same time. You can do this with style sheets by creating a selector like this:

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

This example changes the color *and* centers the text inside an `<h1>` tag. That's why selectors use the funny curly braces (`{` and `}`)—so you can put as many formatting instructions inside them as you want. Each rule is separated from the next using a semicolon (`;`). It's up to you whether you want to include a semicolon at the end of the last rule. Although it's not necessary, Webheads often include one so that it's easy to add another formatting property onto the end of the selector when needed.

---

**Tip:** As in an HTML file, CSS files let you use spacing and line breaks pretty much wherever you want. However, people usually put each formatting instruction on a separate line (as in the example above) in order to make style sheets easy to read.

---

Conversely, you might want to create a formatting instruction that affects several different tags. For example, imagine you want to make sure the first three heading levels all use blue formatting. Rather than writing three separate rules, you can create a selector that includes all three tags, separated by commas. Here's an example:

```
h1, h2, h3 { color: blue }
```

Believe it or not, selectors, properties, and values are the essence of CSS. Once you understand these three ingredients, you're on your way to style sheet gurudom.

Here are a few side effects of the style sheet system that you might not have realized yet:

- A single rule can format a whole whack of HTML. With the selectors you're considering right now (called *type selectors*), every part of your page that uses the tag is affected.
- It's up to you how much you format. You can choose to fine-tune every HTML tag in your Web page, or you can write rules that affect only a single tag, using the techniques you'll see at the end of this chapter (page 173).
- You can create two different rules for the same element. For example, you could create a rule that changes the font of every heading level (`<h1>`, `<h2>`, `<h3>`, and so on), and then add another rule that changes the color of just `<h1>` elements. Just make sure you don't try to set the same property multiple times with conflicting values, or the results will be more difficult to predict.
- Some tags have built-in style rules. For example, text in a `<b>` tag is always displayed with bold text, and text in an `<h1>` heading is always displayed in a large font. But you can override any or all of these rules using style rules. For example, you could explicitly set the font size of an `<h1>` heading so that it appears *smaller* than the following text. Similarly, you can take the underline off of a link.

Don't worry about learning the specific properties and values yet. Later in this chapter, after you see how style sheets work, you'll get acquainted with all the different types of formatting instructions you can use.

## Applying a Style Sheet

Now it's time to see style sheets in action. Before you go any further, dig up the *resume.htm* file you worked on in Chapter 2 (it's also available from the "Missing CD" page at [www.missingmanuals.com](http://www.missingmanuals.com)). You'll use it to test out a new style sheet. Just follow these steps:

1. **First, create the style sheet. You can do this by creating a new file in any text editor like Notepad or TextEdit.**

Creating a style sheet is no different than creating an HTML page—it's all text. Many HTML editors have built-in support for authoring style sheets (see the box "Creating Style Sheets with HTML Editors" on page 145 for more information).

---

**Note:** Remember, Word is *not* a candidate for Best HTML Editor of the Year. In fact, if you make the mistake of editing an HTML page in Word, you're likely to end up with problems (like invalid characters), so don't do it.

---

2. **Type the following rule into your style sheet:**

```
h1 { color: fuchsia }
```

This rule instructs the browser to display all `<h1>` tags in bright fuchsia lettering.

3. **Save the style sheet with the name *resume.css*.**

Like an HTML document, a style sheet can have just about any file name. However, as a matter of convention, style sheets almost always use the extension `.css`. Make sure you save the style sheet in the same folder as the HTML page.

4. **Next, open the *resume.htm* file.**

If you don't have the *resume.htm* file handy, you can test this style sheet with any HTML file that has at least one `<h1>` tag.

5. **In order to use the style sheet with your HTML file, you need to add a link to the HTML file. This link is a special `<link>` tag, which you must place in the `<head>` section of the Web page. Here's the revised `<head>` section with the `<link>` tag you need to add:**

```
<head>
  <link rel="stylesheet" href="resume.css" type="text/css" >
  <title>Hire Me!</title>
</head>
```

The link tag includes three directions. The *rel* attribute indicates that the link points to a style sheet. The *type* attribute describes how the document is encoded. You should copy both of these attributes exactly as shown above, as

they never change. The *href* attribute is the important bit—it indicates the location of the style sheet. (“href” stands for hypertext reference.) Assuming the style sheet is located in the same folder as the HTML file, all you need to supply is the file name. (If the files were located in different folders you’d need to specify the location of the .css file using a file path notation system that you’ll learn about on page 215.)

## 6. Save the HTML file, and open it in a browser.

Here’s what happens. The browser begins processing the HTML document and finds the `<link>` attribute, which tells the browser to find the associated style sheet and apply all its rules. The browser then reads the first (and only) rule in the style sheet. In order to apply this rule, it starts by analyzing the selector, which targets all `<h1>` tags. The browser then finds all the `<h1>` tags, and applies the fuchsia formatting.

The style sheet in this example isn’t terribly impressive. In fact, it probably seems like a lot of work to get a simple pink heading. However, once you’ve got this basic model in place, you can quickly take advantage of it. For example, you could edit the style sheet to change the color. Or, you could add new rules to format other parts of the document. Simply save the new style sheet and refresh the page to see the new rules come into effect.

To see this at work, try changing the style sheet so that it has these rules:

```
body {
    font-family: Verdana,Arial,sans-serif;
    font-size: 83%;
}

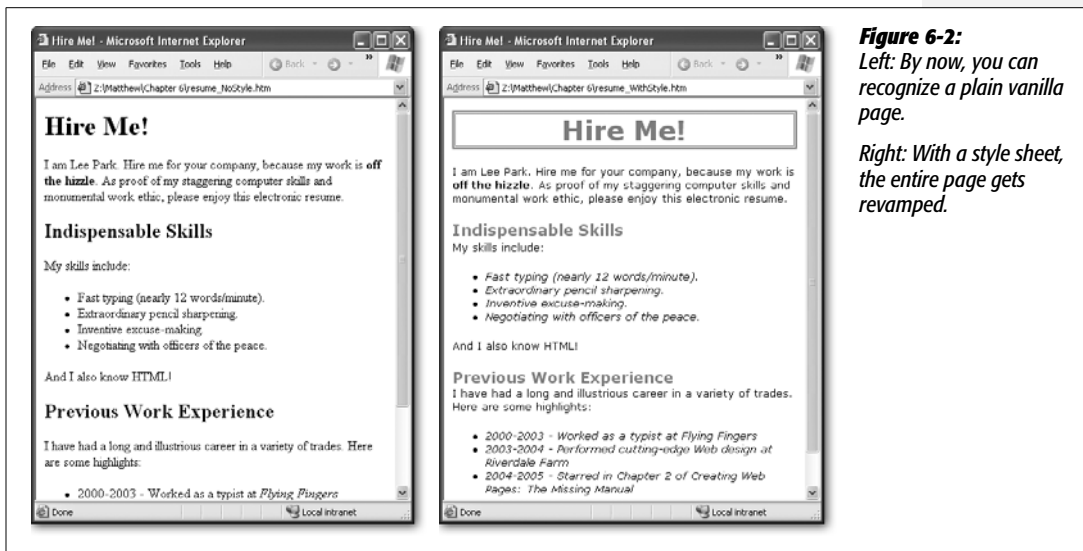
h1 {
    border-style: double;
    color: fuchsia;
    text-align: center;
}

h2 {
    color: fuchsia;
    margin-bottom: 0px;
    font-size: 100%;
}

li {
    font-style: italic;
}

p {
    margin-top: 2px;
}
```

These rules change the font style for the entire document (through the <body> tag), tweak the two heading levels, italicize the list items, and shave off some of the spacing between paragraphs. Although you won't recognize all these rules at first glance, the overall model hasn't changed. Figure 6-2 shows the result.



**Figure 6-2:** Left: By now, you can recognize a plain vanilla page.

Right: With a style sheet, the entire page gets revamped.

## GEM IN THE ROUGH

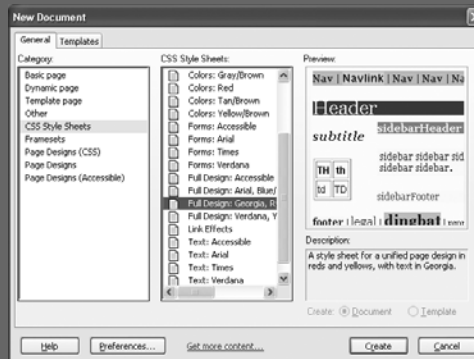
### Creating Style Sheets with HTML Editors

Some HTML editors have handy features for creating style sheets. You won't find this frill in the free Nvu editor, but it does turn up in FrontPage and Dreamweaver.

To create a style sheet in FrontPage, choose File → New to show the “New page” task window on the right. Then, click the “More page templates” link to show the Page Templates dialog box. Choose the Templates tab (which is confusingly called the Style Sheets tab in some older versions of FrontPage). You'll see a list of predefined templates—choose one of these and you'll start off with a pile of color-coordinated rules you can apply to a Web page right away. (And of course, you're free to change or replace the rules as you see fit.)

Dreamweaver goes one better than FrontPage. It also includes a list of prebuilt style sheets—to see it, choose File → New and then select the Templates tab in the New Document dialog box (shown in this illustration). Not only will

you see a list of style sheets, but you'll also see a small preview window that shows you a sample of what the rules in the style sheet look like.



## Internal style sheets

The previous example demonstrated an external style sheet. External style sheets are everybody's favorite way to use CSS, because they allow you to link a single lovingly crafted style sheet to as many Web pages as you want. However, sometimes you don't have an entire Web site in mind, and you'd be happy with a solution that's a little less ambitious.

An internal style sheet is a style sheet that's not linked, but instead is *embedded* in the <head> area of a Web page. Yes, it bulks up your Web pages and it forces you to give each Web page a separate style sheet. But occasionally, the convenience of having just one page with its own style rules makes it worthwhile.

To change the earlier example so that it uses an internal style sheet, remove the <link> element and add the style rules in a <style> element inside the <head> section of the Web page, as shown here:

```
<head>
  <style type="text/css">
    h1 { color: fuchsia }
  </style>
</head>
```

To ensure optimum compatibility with really old browsers, there's another trick Web gurus often use—they put the style information in between comment tags:

```
<head>
  <style type="text/css">
  <!--
    h1 { color: fuchsia }
  -->
  </style>
</head>
```

This way, ancient browsers that have no understanding of style sheets (like Netscape 3), won't inadvertently display the style information in the Web page because it's hidden in the page. (On the other hand, browsers that are CSS-conversant see the instructions just fine.) Of course, you're unlikely to run into this problem these days, but many Web authors still follow this best practice.

## Inline styles

If you want to avoid writing a style sheet altogether, there's another approach you can use. Inline styles allow you to insert the property and value portion of a style sheet rule right into an HTML tag. (You don't need the selector because it's obvious that you want to format the current tag.) For example, here's how you format a single heading with an inline style:

```
<h1 style="color: fuchsia">Hire Me!</h1>
```

When you use this approach, it affects only the tag where the style rule is placed. If there are other `<h1>` headings in the Web page, they aren't affected.

## WORD TO THE WISE

## Boosting Style Sheet Speed

External style sheets are more efficient in Web sites because of the way that browsers use *caching*. (Caching is a performance-improving technique where browsers store a copy of some downloaded information on the Web surfer's computer, so they don't need to download it again.)

When you link to a style sheet, the browser makes a separate request to get that style sheet file, as shown back in Figure 6-1. However, if you have another Web page that requests the *same* style sheet, the browser is intelligent enough to realize it already has the right style sheet on hand. As a result, it doesn't make the request. Instead, it just

reuses the cached copy of the style sheet, which makes the page load a little bit faster. (Of course, browsers only cache things for so long. If you surf to the same site tomorrow, the browser will re-request the style sheet in case it's changed.)

If you put the style sheet into each Web page, the browser always downloads the full Web page with the duplicate copy of the style sheet. It has no way to realize that you're reusing the same rules. Although this probably won't make a huge difference, in a Web site with lots of pages it could start to add up. Speed is just one more reason Web veterans prefer external style sheets.

Inline styles may seem appealing at first glance, because they're clear and straightforward. You define the formatting information exactly where you want to use it. However, if you try to format a whole Web page this way, you'll realize why Web gurus universally shun them. Quite simply, the average CSS formatting rule is long. If you need to put it inside an HTML page alongside your content, and copy it each time you use the tag, you'll quickly end up with a Web page that's mangled beyond all recognition. For example, consider a more complex heading that needs several style rules:

```
<h1 style="border-style: double; color: fuchsia; text-align: center">Hire  
Me!</h1>
```

Even if this occurs only once in the document, it's already becoming a loose and baggy monstrosity. Try to avoid inline styles if you can.

---

**Note:** Novice Web designers often get into trouble with inline styles when they use WYSIWYG editors like FrontPage to format their Web pages. Every time you change a formatting detail in FrontPage (like the color, alignment, and so on), it quietly adds a dollop of style information to your HTML. That makes these files quite difficult to edit (and gives FrontPage a bit of a bad reputation). Now that you know better, you'll be able to use a WYSIWIG editor *and* styles to make sure your HTML stays neat and tidy.

---

## The Cascade

By now, you might be wondering what the “cascading” part of “cascading style sheets” means. It refers to the way the browser decides which rules take precedence in case you've got multiple sets of rules.

For example, if you indicate that `<h1>` headings should have blue letters using an external style sheet, and then apply bold formatting with an inline style, you'll end up with the sum of both changes: a bold blue-letter heading. However, it's not as clear what happens if the rules conflict—for example, if one rule specifies blue text while another mandates red. Which color setting wins?

To determine the answer, you need to consult the following list to find out which rule has highest priority. This list indicates the steps the browser follows when applying styles. The steps toward the bottom are the most powerful: they're performed after the steps at the top, and so they overwrite any earlier formatting.

1. Browser default
2. External style sheet
3. Internal style sheet (inside the `<head>` tag)
4. Inline style (inside an HTML element)

So, if an external style sheet property conflicts with an internal style sheet, the internal style sheet wins.

Based on this behavior, you might think that you can use this cascading behavior to your advantage by defining general rules in external style sheets, and overriding them with the occasional exception using inline styles. However, there's actually a much better option. Rather than formatting every matching tag, you can specifically format individual tags by using class selectors (see page 173 for details).

---

**Note:** The “cascading” in cascading style sheets is a little misleading, because in most cases you won't use more than one type of style sheet (for the simple reason that it can quickly get confusing). Most Web artists favor external style sheets primarily and exclusively.

---

## Inheritance

Along with the idea of cascading styles, there's another closely related concept—style *inheritance*. In order to understand style inheritance, you need to remember that in HTML documents, one tag can contain other tags. For example, the `<ul>` (unordered list) tag contains `<li>` (list item) tags. Similarly, a `<p>` paragraph tag can contain character formatting tags like `<b>` and `<i>`, and the `<body>` tag contains the whole document.

Thanks to inheritance, when you apply formatting to an element that contains *other* elements, the rule is applied to *everything*. For example, if you set the `<body>` element to use a specific font (as in the résumé style sheet shown earlier), the font applies to every element inside the `<body>` element, including all the headings, paragraphs, lists, and so on.

---

**Note:** There are some style properties that break the rules (for example, margin settings are never inherited) but most don't. Look for the Can Be Inherited column in each table in this chapter to figure out whether a property will be inherited.

---

However, there's a trick. Sometimes, formatting rules may overlap. In this case, the most specific rule—that is, the one closest to the tag—wins. For example, settings in an `<h1>` element override settings in a `<body>` element for all level 1 headings. Or consider this style sheet:

```
body {
  color: black;
  text-align: center;
}

ul {
  color: fuchsia;
  font-style: italic;
}

li {
  color: red;
  font-weight: bold;
}
```

These rules can overlap. In a typical document (see Figure 6-3) a `<li>` (list item) is placed inside a list tag like `<ul>`, which in turn exists inside the `<body>` tag.

Crafty style sheet designers can use this behavior to their advantage. For example, you might apply a font to the `<body>` element so that everything in your Web page—headings, paragraph text, lists, and so on—has the same font. Then, you can judiciously override this font for a few specific tags by applying more formatting rules.

Although you probably won't see cascading styles in action very often, you'll almost certainly use style inheritance.

---

**Note:** Now that you've learned how style sheets work, you're ready to start with the hard part—learning about the dozens of different formatting properties you can change. In this chapter, you won't learn about every property. For example, there are some properties that apply primarily to pictures and tables. You'll learn about these properties in later chapters.

---

## Colors

It isn't difficult to inject some color into a Web page. There are really just two color-related properties that you'll use, and they're listed in Table 6-2.

**Table 6-2.** Color Properties

Property	Description	Common Values	Oldest Supported Browsers	Can Be Inherited
color	The color of the text. This is a handy way to make headings or emphasized text stand out.	A color name, HTML color code, or RGB color value.	IE 3, Netscape 4	Yes
background-color	The color behind the text, for just that tag.	A color name, HTML color code, or RGB color value. You can also use transparent.	IE 4, Netscape 4	No <sup>a</sup>

<sup>a</sup> The background-color isn't inherited (page 148), which means a tag doesn't get the same background color as the tag that contains it. However, there's a trick. If you don't explicitly assign a background color to a tag, its color is transparent. That means the color of the containing tag will still show through, which has the same effect.

The *color* property is easy to understand—it's the color of your text. The *background-color* property is a little more unusual. If you apply a background color to the <body> tag, the whole Web page is affected, as you might expect. However, if you use the background color on an individual element, the results are a bit stranger. In CSS, what's inside of each tag is treated as though it exists in an invisible rectangle.

When you apply the background color to an element, the color applies just to this rectangle.

For example, the following style sheet applies different background colors to the page, headings, paragraphs, and any bold text:

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

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

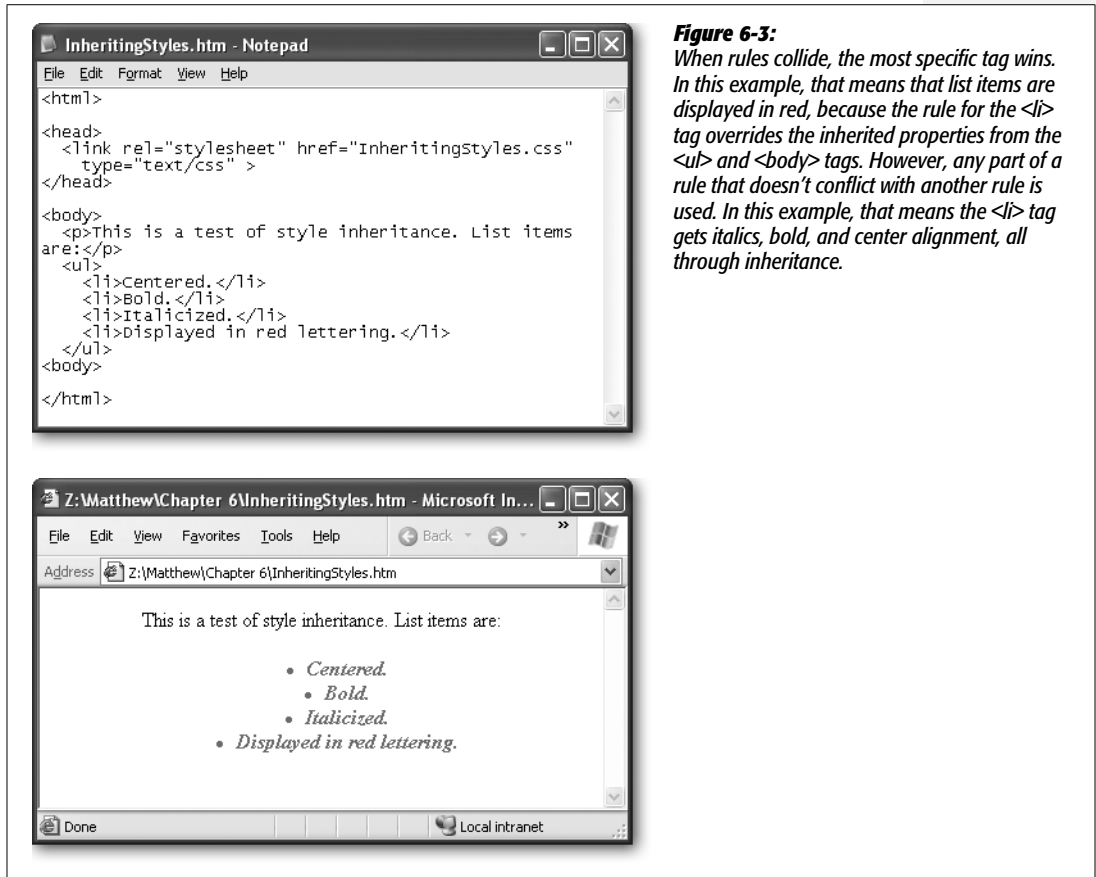
```

p {
  background-color: lime;
}

b {
  background-color: white;
}

```

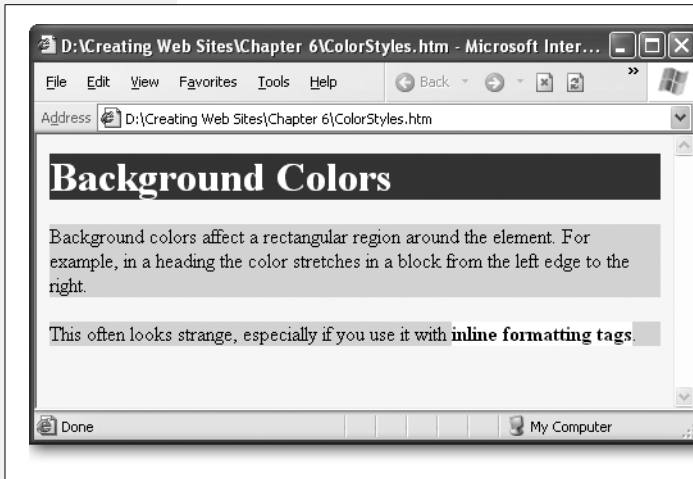
Figure 6-4 shows the result.



## Specifying a Color

The trick to using colors is finding the appropriate code that indicates the exact shade of electric blue you love. You have several ways to go about this. First of all, you can indicate your color of choice with a plain English name, as you've seen in the examples so far. Unfortunately, this system only works with a small set of 16

color names (aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, purple, red, silver, teal, white, and yellow). Some browsers accept other names, but none are guaranteed to be widely supported, so it's best to use another approach. CSS gives you two more options: hexadecimal (or HTML) color values, and RGB (or red-green-blue) values.



**Figure 6-4:**

*If you apply a background color to a tag like `<h1>`, it colors just that line. If you use it on an inline tag like `<b>` or `<span>`, it affects only the words in the tag. Both results look odd—it's a little like someone went wild with a highlighter. A better choice is to apply the background color to the whole page by applying it to the `<body>` tag, or just to a large box-like portion, using a tag like `<div>`.*

### **Hexadecimal color values**

With *hexadecimal color values* you use a strange-looking code with a pound sign (#) at the beginning. Technically, hexadecimal colors are made up of three numbers representing the red, green, and blue component of the color (similar to the RGB colors you'll learn about in the next section). However, these colors have been combined in a manner that's perfectly understandable to computers, but utterly baroque to normal people. You'll find hexadecimal color notation kicking around the Web a lot, because it's the original format that was designed for HTML. However, it's about as intuitive as reading the ones and zeroes that power your computer.

Here's an example:

```
body {
    background-color: #E0E0E0
}
```

Even a computer nerd can't tell that this applies a light gray background. (See "Finding the Right Color" for some tips on how to pick out hexadecimal color values, if you're determined to use this system.)

### **RGB color values**

The other approach to specifying color values is to use RGB (or *red-green-blue*) values. According to this more logical approach, you simply specify how much of

each color you want to “mix in” to create your final color. Each component takes a number from 0 to 255. For example, a color that’s composed of red, green, and blue, each set to 0, appears white; on the other hand, all those values set to 255 generates black.

Here’s an example of a nice lime color:

```
body {
  background-color: rgb(177,255,20)
}
```

#### FREQUENTLY ASKED QUESTION

### Web-Safe Colors

*Will the colors I pick show up on other computers?*

When colors began appearing as the latest fad in Web pages, the computing world was very different. The average computer couldn’t handle a really wide variety of colors. Many computers were limited to a relatively small set of 256 colors, and had to deal with other colors by *dithering* (a dubious process that combines little dots of several colors to simulate a different color, leading to an unattractive speckled effect). To avoid dithering, Web designers came up with a standard called *Web-safe colors*, which identifies a set of 216 colors that can be reliably used on any computing platform. Even better, they always look almost exactly the same.

Today, the world is a little different, and you’d be hard pressed to find a computer that can’t display at least 16,000 colors (a standard called 16-bit color, or *high color*). Most support a staggering 16.7 million (a standard called 32-bit color, or *true color*). In this environment, it’s rarely worth worrying about the Web-safe colors anymore, and the standard is just another piece of computing history.

However, there is one exception. If you plan to create Web pages for very small devices (like cell phones or palmtop computers), which have much leaner hardware, you might change your mind and decide to pare down your Web pages and limit yourself to Web-safe colors only. To check out the list of safe colors, surf to [www.w3schools.com/css/css\\_colors.asp](http://www.w3schools.com/css/css_colors.asp).

Even if you aren’t concerned about serving this still relatively small audience of tiny devices, it’s still a good idea to look at your Web pages on a variety of computers. That’s because different monitors don’t always display the same colors—some tend to tint colors unexpectedly, and Windows computers tend to produce darker colors than their Macintosh counterparts (even when using the same monitor). Pick colors carefully, because a color combination that looks great on your computer can look nauseating (or worse, be illegible) on someone else’s.

### Finding the Right Color

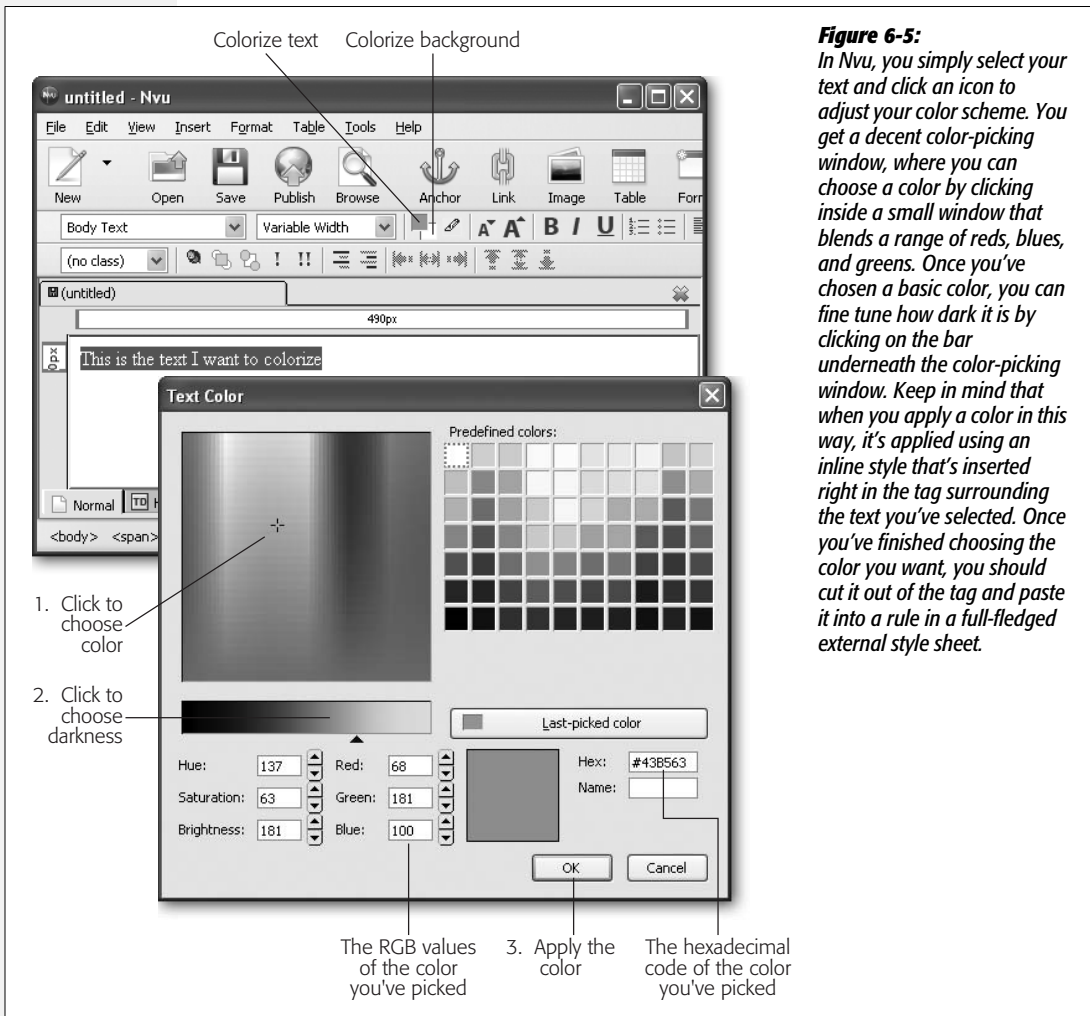
Style sheets can handle absolutely any color you can imagine. But how do you find the color code for the perfect shade of sunset orange (or dead salmon) that you need?

Sadly, there’s no way this black and white book can show you your choices. But there are a great number of excellent color-picking programs online. You have two options here—you can download a free color-picking program, or you can hunt around on the Web to find an online color-picking program, which is often more convenient.

For example, try [www.webtemplates.com/colors](http://www.webtemplates.com/colors), where all you need to do is click a picture to preview the color you want (and see its hexadecimal code). Other handy online color pickers include <http://mediagods.com/tools/rgb2hex.htm> and [www.colorschemer.com/online.html](http://www.colorschemer.com/online.html) (where you can see groups of colors that match).

**Note:** The RGB system lets you pick any of 16.7 million colors, which means that none of these Web sites will actually show you every single possible RGB color code (if they do, make sure you don't hit print; even with ten colors per line, you'd wind up with thousands of pages). Instead, most sites limit you to a representative sampling of colors. This works, because many colors are so similar that they're nearly impossible to distinguish by eye.

If you're using an HTML editor like FrontPage, Dreamweaver, or Nvu, life gets a little easier. These programs have built-in color picking tools (see Figure 6-5).



**Tip:** The RGB color standard is alive and well in many computer programs. For example, if you see a color you love in a professional drawing program, odds are there's a way to get the red, green, and blue values for that color, which you can then use in a style sheet. This gives you a great way to match the text in your Web page with a color in a picture. Now that's a trick that pleases the strictest interior designer!

## DESIGN TIME

### Making Color Look Good

Nothing beats black on white for creating a crisp, clean, easy-to-read Web page with real presence. Black text and white backgrounds also work best in Web pages that have a lot of colorful pictures. It's no accident that almost every top Web site, from news sites ([www.cnn.com](http://www.cnn.com)) to search engines ([www.google.com](http://www.google.com)) to e-commerce shops ([www.amazon.com](http://www.amazon.com)) and auction houses ([www.ebay.com](http://www.ebay.com)) all use the winning combination of black on white.

But what if you're just too colorful a person to leave your Web page in plain black and white? The best advice is to follow the golden rule of color: *use restraint*. Unless you're creating a sixties revival site or a Led Zeppelin tribute page, you don't want color to run wild. Here are some ways that you can inject a splash of color without letting it take over your Web page:

- **Go monochrome.** That means use black, white, and one other dark color. For example, Time magazine's Web site ([www.time.com](http://www.time.com)) uses its familiar bold red color for headlines.
- **Use a lightly shaded background.** Sometimes, a faint wash of color in the background is all you need to perk up a site. For example, a gentle tan or gold

can suggest elegance or sophistication (see the Harvard library at <http://lib.harvard.edu>). Or, light pinks and yellows can get shoppers ready to buy sleepwear and other feminine accoutrements (see Victoria's Secret at [www.victoriasecret.com](http://www.victoriasecret.com)).

- **Use color in a box.** Shaded boxes are a frequently used technique to highlight important areas of a Web page (see Wikipedia at <http://en.wikipedia.org>). You'll learn how to create boxes later in this chapter on page 169.
- **Be careful about using white text.** White text on a black or dark blue background can be striking—and strikingly hard to read. The rule of thumb is avoid it, unless you're trying to make your Web site seem futuristic, alternative, or gloomy. (And even if you do fall into one of these categories, you might still get a stronger effect with a white background and a few well-chosen graphics with splashy electric colors.)

## Fonts

Using the CSS font properties, you can choose a font family, the font weight (its boldness setting), and font size (see Table 6-3). Be prepared, however, for a bit of Web-style uncertainty; this is one case where life isn't as easy as it seems.

The inescapable problem you face when using CSS font properties is that no two computers have the same set of fonts. A simple way to solve this would be to create Web browsers that could automatically download new fonts they don't have—but this would be a Web nightmare. First, it could swamp the average computer's hard drive with thousands of (potentially low-quality) fonts. Second, it would

infuriate the software companies who sell fonts. (Fonts aren't free, and so copying them wantonly from one computer to another isn't kosher.)

**Table 6-3.** *Font Properties*

Property	Description	Common Values	Oldest Supported Browsers	Can Be Inherited
font-family	A list of font names. The browser scans through the list until it finds a font that's on the browser's PC. If no supported font is found, it uses the standard font it always uses.	A font name (like Verdana, Times, or Arial) or one of the generic family names: serif sans-serif monospace	IE 3, Netscape 4	Yes
font-size	Sets the size of the font.	A specific size, or one of these values: xx-small x-small small medium large x-large xx-large smaller larger	IE 3, Netscape 4	Yes
font-weight	Sets the weight of the font (how bold it appears).	normal bold bolder lighter	IE 4, Netscape 4	Yes
font-style	Lets you apply italic formatting.	normal italic	IE 4, Netscape 4	Yes
font-variant	Lets you apply small caps, which turns lowercase letters into smaller capitals (LIKE THIS).	normal small-caps	IE 4, Netscape 6	Yes
text-decoration	Applies a few miscellaneous text changes, like underlining and strikeout. Technically speaking, these aren't part of the font (they're added in by the browser).	none underline overline line-through	IE 4, Netscape 4	Yes
text-transform	Transforms text so that it's all capitals or all lowercase.	none uppercase lowercase	IE 4, Netscape 4	Yes

There may be practical solutions to these problems, but unfortunately, browser companies and the people who make Web standards have never agreed on any. As a result, any font settings you specify are just recommendations. If a browser doesn't have the font you request, it reverts to the standard font that the browser uses whenever it's on a site that doesn't have special font instructions.

Given that caveat, you're probably wondering why you should bother to configure font choices at all. Well, here's one bit of good news. Instead of requesting a font and blindly hoping that the browser has it, you can create a list of *font preferences*. That way, the browser will try to match your first choice and, if that fails, your second choice, and so on. At the end of this list, you should use one of the few standard fonts that almost all platforms are known to support in some variation. You'll see this technique at work in the next section.

DESIGN TIME

### Graphical Text

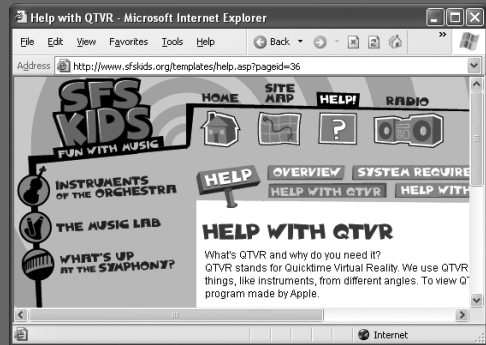
The only guaranteed cure for font woes is *graphical text*. With graphical text, you don't type your content in an HTML file. Instead, you perfect it in a drawing program, and then save it as a picture. Finally, you display the picture of the text in your page using the `<img>` tag.

Graphical text is clearly unsuitable for large amounts of text. First of all, it bloats the size of your Web page horribly. It's also much less flexible. For example, graphical text can't adjust itself to fit the width of the browser window or take into account your visitors' browser preference settings. There's also no way for a Web surfer to search through your page hunting for specific words (or for a Web search engine to figure out what's on your Web site).

However, graphical text is commonly used for menus, buttons, and headings, where these issues aren't nearly as important. You'll find this technique used on countless Web sites. For just one example, look at the children's site in this illustration. There's only a little real text here—the distinctive navigation buttons and headings are all graphics.

Often, graphical text isn't as obvious. For example, you may have never noticed that the section headings on your favorite online newspaper are actually images. To figure out if a Web site is using graphical text or the real deal, try to select the text. If you can't, the text is really a picture.

You'll learn how to use graphics (including graphical text) in Chapter 7.

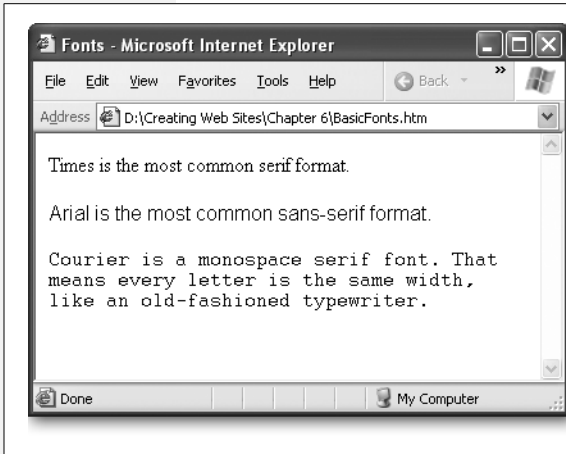


### Specifying a Font

To select a font, you use the *font-family* attribute. Here's an example that changes the font for an entire page:

```
body {
    font-family: Arial;
}
```

Arial is a *sans-serif* font that's found on just about every modern computer, including those that run the Windows, Mac, Unix, and Linux operating systems. (See Figure 6-6 for more about the difference between serif and sans-serif.)



**Figure 6-6:** *Serif fonts use adornments, or serifs, that make them easier to read in print. This book is written using a serif font. If you look closely at the letter “S,” you’ll see tiny curlicues in the top-right and bottom-left corners. On the other hand, sans-serif fonts have a spare, streamlined look. Depending on the font, they can seem less bookish, less formal, more modern, and colder.*

To be safe, when you specify a font, you should always use a font list that ends with a generic font family name. Every browser supports generic family names, which include serif, sans-serif, and monospace.

Here’s the modified rule:

```
body {
    font-family: Arial, sans-serif;
}
```

---

**Tip:** If your font has a space in the name, make sure you enclose the whole font name in quotations.

---

At this point, you might be tempted to get a little creative with this rule by adding support for a less common sans-serif format. Here’s an example:

```
body {
    font-family: Eras, Arial, sans-serif;
}
```

If Eras is relatively similar to Arial, this technique might not be a problem. But if it’s significantly different, this is a bad idea.

The first problem is that by using on a non-standard font, you’re creating a Web page whose appearance may vary dramatically depending on the fonts on the Web surfer’s computer. Whenever pages vary, it becomes more difficult to really tweak them to perfection, because you don’t know exactly how they’ll be displayed. Different fonts take up different amounts of space, and if text grows or shrinks, the

layout of other elements (like pictures) changes, too. Besides, is it really that pleasant to read KidzzFunScript or SnoopDawg font for long periods of time?

A more insidious problem occurs if another browser has a font with the same name that looks completely different. Even worse, browsers may access an online database of fonts to try and find a similar font that is installed on the Web surfer's computer. This approach can quickly get ugly. At worst, either of these problems can lead to illegible text.

---

**Note:** Most HTML editors won't warn you when you apply a non-standard font. So be on your guard. If your font isn't one of a small set of widely distributed Web fonts (more on which those are in a moment), you shouldn't try to use it.

---

## Finding the Right Font

To make sure your Web page displays correctly, you should use a standard font that's widely available. But just what are these standard fonts? Unfortunately, Web gurus don't completely agree.

But if you want to be really conservative, you won't go wrong with any of these fonts:

- Times
- Arial and Helvetica
- Courier

Of course, all of these fonts are insanely boring. If you want to take on more risk, you can use one of the following fonts, which are found on almost all Windows and Mac computers (but not necessarily on other operating systems like Unix):

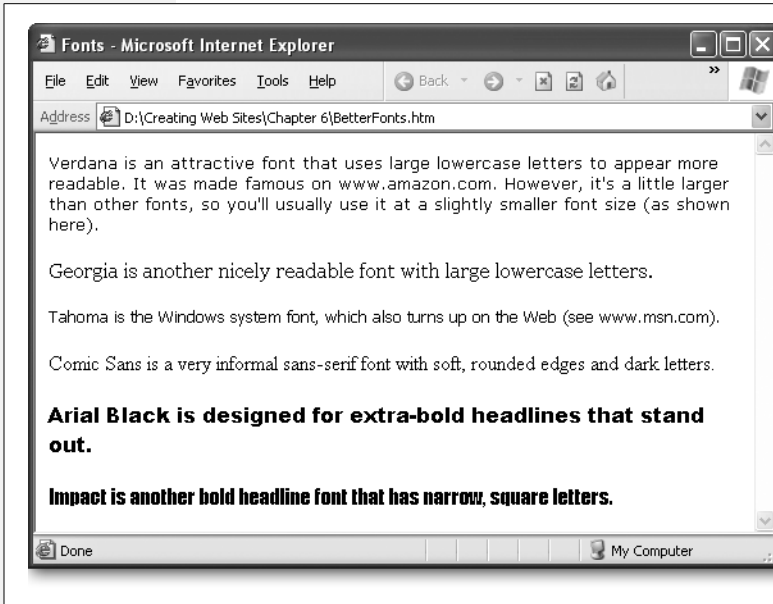
- Verdana
- Georgia
- Tahoma
- Comic Sans
- Arial Black
- Impact

To compare these different fonts, see Figure 6-7.

Verdana, Georgia, and Tahoma can all help give your Web pages a more modern look. However, Verdana and Tahoma usually need to be ratcheted down one notch in size (a technique described in the next section).

For good resources that discuss different fonts, what platforms reliably support them, and the pros and cons of each font family (for example, some fonts look nice

on screen but generate lousy printouts) see <http://web.mit.edu/jmorzins/www/fonts.html> and [www.upsdell.com/BrowserNews/res\\_fontsamp.htm](http://www.upsdell.com/BrowserNews/res_fontsamp.htm).



**Figure 6-7:**  
*Have you spotted these fonts at large on the Web?*

## Font Sizes

Once you've sorted out the thorny issue of choosing your font, you may also want to change the size. It's important that you select a text size that's readable and looks good. Resist the urge to shrink or enlarge text to suit your personal preferences. Instead, aim to match the standard text size that you see on other popular Web sites.

Despite what you might expect, you don't have complete control over the font size in your Web pages. Most Web surfers use browsers that let them scale font sizes up or down, either to fit more text on screen or (more commonly) to make text easy to read on a high-resolution monitor. In Internet Explorer and FireFox, you'll find these options in the View → Text Size menu.

The browser's font size settings don't completely override the size you've set in your Web page. Instead, they just tweak it up or down. For example, if you choose to use a large font size (which corresponds to a setting of about 15 points in a word processor) and an Internet Explorer surfer selects View → Text Size → Larger, the text size grows about 20 percent in size (to 18 points).

The fact that your visitors have this kind of control is another reason you shouldn't use particularly small or large font sizes in your Web pages. When they're combined with the browser preferences, a size that's a little on the large size could become gargantuan, and text that's slightly small could turn unreadable. The best

defense for these problems is to test out your Web page with different browsers *and* different font size preferences.

As you'll discover in the following sections, you have three choices for setting font sizes.

---

**Tip:** Getting the right font size is trickier than you might think, because different browsers will interpret your font sizes differently. If you want to explore Web typography in even more detail, check out the incredibly in-depth information that's available at <http://usabletype.com/css>. It's somewhat technical, but remarkably thorough.

---

### ***Absolute sizes (keywords and percentages)***

The simplest approach for specifying the size of your text is to use one of the size values listed in Table 6-3. For example, to create a really big heading and ridiculously small text, you could use these two rules:

```
body {
    font-size: xx-small;
}
h1 {
    font-size: xx-large;
}
```

These size keywords are often called *absolute sizes*, because they apply an exact size. Exactly what size, you ask? Well, that's where it gets a bit complicated. These details actually aren't set in stone—different browsers are free to interpret them in different ways. The basic rule of thumb is that the font size *medium* corresponds to the standard text size that the browser uses. Every time you go up a size level, you add about 20 percent in size. (For math geeks, that means every time you go down a size you lose about 17 percent.)

A typical standard font size for most computers is 12 points (although text at this size typically appears smaller on Mac computers than on Windows computers). That means *large* text is approximately 15 points, *x-large* text is 18 points, and *xx-large* text is 27 points.

Figure 6-8 shows the basic sizes you can choose from.

There's one serious drawback to using the size keywords—they really aren't absolute. As described above, when you set a font to *medium* you're supposed to get a browser's standard text size. Unfortunately, that's not how Internet Explorer sees it. Instead, IE displays its standard text size when it sees font that's set to *small*. That means if you want to get a little smaller (which is useful for some large fonts, like Verdana), you actually need to choose *x-small*. Unfortunately, other, more standards-aware browsers (like Firefox) don't have this idiosyncrasy. As a result, pages that look perfect on Internet Explorer are likely to look smaller on Firefox when you use size keywords.

The best solution to correct this problem is to use percentage sizes instead of size keywords. For example, if you want to make sure text is normal size, use this rule:

```
body {
    font-size: 100%;
}
```

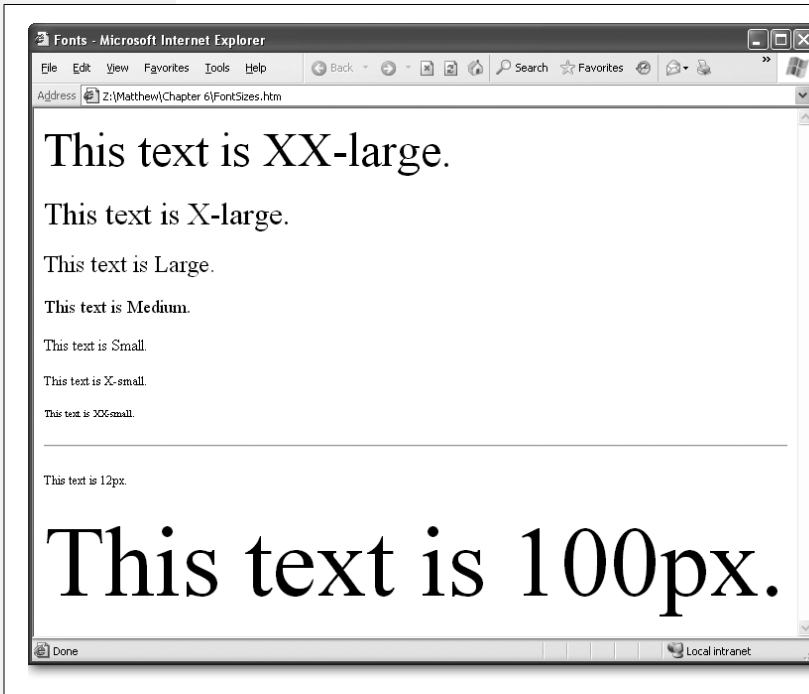
And if you want to make text smaller, use something like this:

```
body {
    font-family: Verdana,Arial,sans-serif;
    font-size: 83%;
}
```

This sets text to be 83 percent of the standard size. It doesn't matter whether the standard size is considered *small* (Internet Explorer) or *medium* (most other browsers). This particular example creates nicely readable text with the Verdana font.

It's just as easy to upsize text:

```
h1 {
    font-size: 120%;
}
```



**Figure 6-8:** There are seven standard sizes, ranging from xx-large to xx-small. You can also pick your own size by specifying a pixel measurement.

But keep in mind that 100 percent always refers to the standard size of normal *paragraph* text (not the standard size of the element you're styling). So if you create a heading with 120 percent sized text, your heading is going to be only a little bigger than normal paragraph text, which is actually quite a bit smaller than the normal size of an `<h1>` heading.

Using percentage sizes is the safest, most reliable way to size text. Not only does it provide consistent results across all browsers, it also works in conjunction with the browser size preferences described earlier.

---

**Tip:** When you use absolute sizes, you create flexible pages. If the visitor ratchets up the text size using his browser's preferences, all your other fonts are resized proportionately.

---

### Relative sizes

Another approach for setting fonts is to use one of two *relative size* values—larger or smaller. This is a bit confusing, because as you just learned in the last section, absolute sizes are already relative—they're all based on the browser setting for standard text.

The difference is that relative sizes are influenced by the font of the element that contains them. The easiest way to understand how this works is to consider the following style sheet, which has two rules:

```
body {
  font-size: xx-small;
}
b {
  font-size: larger;
}
```

The first rule applies an absolute *xx-small* size to the whole page. The second rule (the relative one) *inherits* the *xx-small* size (see page 146 for a recap about inheritance). However, it then steps the font size up one notch to *x-small*.

Now consider what happens if you edit the body style to use a larger font, like this:

```
body {
  font-size: x-small;
}
```

Now all bold text will be shown one level up from *x-small*, which is *small*.

The only limit of the two relative sizes is that they can step up or down only one level. You can get around this limitation by using font numbers. For example, a size of +2 is a relative size that increments a font two levels. Here's an example:

```
body {
  font-size: x-small;
}
```

```
b {  
    font-size: +2;  
}
```

Now the bold text is shown in *medium* text, because *medium* is two levels up from *x-small*.

Relative sizes are a little trickier to get used to than absolute sizes. You're most likely to use them if you have a style sheet that has a lot of different sizes. For example, you might use a relative size for bold text if you want to make sure bold text is always a little bit bigger than the text around it. If you were to use an absolute size instead, the bold text would appear large in relation to a small-sized paragraph, but it wouldn't stand out in a large-sized heading.

### **Exact sizes (pixels)**

Most of the time, you should rely on absolute and relative sizing. However, if you really must have more control, you can customize your font size precisely by specifying a *pixel size*. Pixel sizes can range wildly, with 12px and 14px being about normal for body text. To specify a pixel size, use the number immediately followed by the letters *px*, as shown here:

```
body {  
    font-size: 11px;  
}  
h1 {  
    font-size: 24px;  
}
```

---

**Tip:** Don't put a space between the number and the letters px. If you do, your rule may work in Internet Explorer but thoroughly confuse other browsers.

---

As always, you need to test, refine, and retest to get the right sizes. Some fonts look bigger than others, and require smaller sizes. Other fonts work well at larger sizes, but become less legible as you scale them down in size.

Web purists avoid using exact sizes because they are horribly inflexible on Internet Explorer. For example, if a near-sighted surfer has upped the text size settings in Internet Explorer, it won't have any effect on your page. (For some reason, other browsers don't suffer from this problem—they're able to resize pages even if you use pixel sizes.) As a result, when you use pixel sizes you could inadvertently lock out certain audiences or create pages that are difficult to read or navigate on certain types of browsers. It just goes to show that in the Web world there's a price to be paid for getting complete control over formatting.

## Text Alignment and Spacing

CSS includes a great many properties for controlling how text appears on a Web page. If you've ever wondered how to indent paragraphs, space out lines, or center a title, these are the tools you need.

Table 6-4 has the details on all your alignment options.

**Table 6-4.** *Alignment and Spacing Properties*

Property	Description	Common Values	Oldest Supported Browsers	Can Be Inherited
text-align	Lines the text up on one or both edges of the page.	left right center justify	IE 4, Netscape 4	Yes
text-indent	Indents the first line of text (typically in a paragraph).	A pixel value (indicating the amount to indent) or percentage of the width of the containing tag.	IE 4, Netscape 4	Yes
margin	Sets the spacing that's added around the outside of a block element (page 116). You can also use the similar properties margin-bottom, margin-left, margin-right, and margin-top to change the margin on just one side.	A pixel value or percentage indicating the amount of space to add around the element.	IE 4, Netscape 4	No
padding	Sets the spacing that's added around the inside of a block element. Has the same effect as margin, unless you have an element with a border or background color (see page 151 for more).	A pixel value or percentage indicating the amount of space to add around the element.	IE 4, Netscape 4	No
word-spacing	Sets the space between words.	A pixel value or percentage.	IE 6, Netscape 6	Yes
letter-spacing	Sets the space between letters.	A pixel value or percentage.	IE 6, Netscape 6	Yes

**Table 6-4.** Alignment and Spacing Properties (continued)

Property	Description	Common Values	Oldest Supported Browsers	Can Be Inherited
line-height	Sets the space between lines.	A pixel value or percentage. You can also use a multiple (i.e., use 2 for double-spacing).	IE 4, Netscape 4	Yes

For example, if you want to create a page that has indented paragraphs (like a novel or newspaper), use this style sheet rule:

```
p {
  text-indent: 20px
}
```

In the following sections, you'll see examples that use the alignment and margin properties.

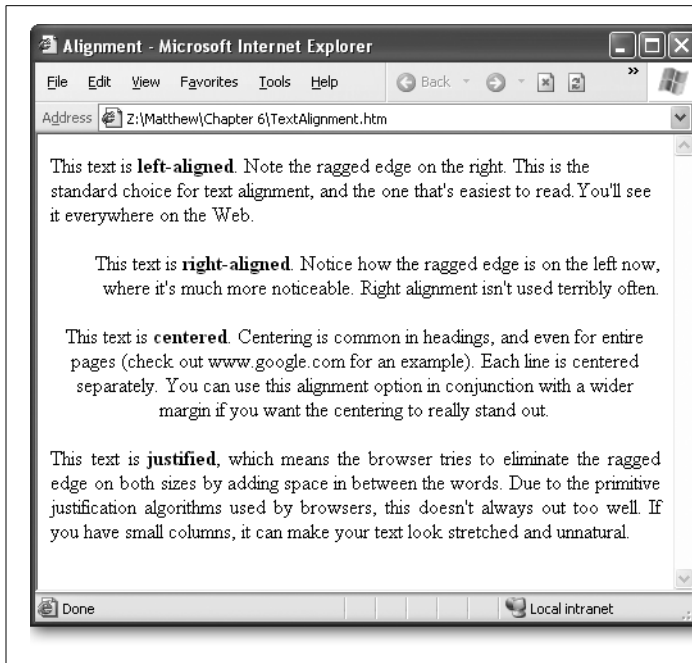
## Alignment

Ordinarily, all text in a Web page is lined up on the left side of the browser window. Using the *text-align* property, you can center text, line it up on the right edge, or justify it. Figure 6-9 shows your options.

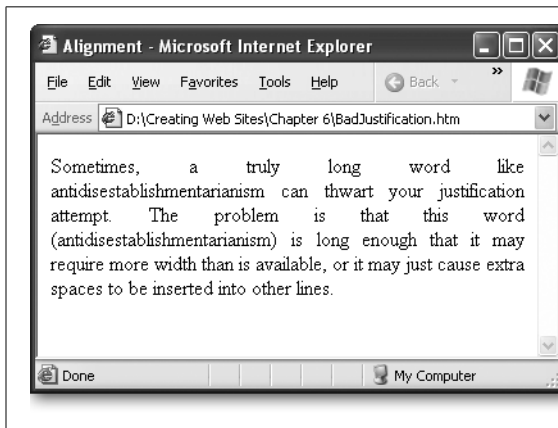
The most interesting choice is full justification, which tries to line text up so it lines up on both sides. Full justification, which you get by using the *justify* setting, is common in print (including books like this one). Originally, printers preferred full justification because it helps cram more words into each page (thereby reducing the number of pages and the printing cost). These days, it's a way of life. Many people feel text with full justification looks neater and cleaner than text with a ragged edge, even though tests show plain, unjustified text is easier to read.

Justification doesn't work as well in the Web world as in print. A key problem is the lack of word-splitting rules, which allow long words to be divided in a printed page. The method browsers use to justify text is relatively simplistic. Essentially, the browser adds words one at a time to a line, until no more words can fit, at which point spacing is added between the words to pad it to full length. By comparison, the best page layout systems for print can analyze an entire paragraph as a whole, and find the optimum justification strategy that best satisfies every line. In problematic cases, a skilled typesetter may need to step in and adjust the line

breaking manually. Compared to this approach, Web browsers are irredeemably primitive, as you can see in Figure 6-10.



**Figure 6-9:**  
This page shows common types of text alignment.



**Figure 6-10:**  
If you decide to use full justification in a Web page, make sure your text lines are fairly long. Otherwise, you'll quickly wind up with gaps and rivers of white space. Few Web sites use justification.

## Spacing

To adjust the spacing around any element, you use the *margin* property. For example, here's a rule that adds a fixed spacing of eight pixels to all sides of a paragraph:

```
p {
  margin: 8px;
}
```

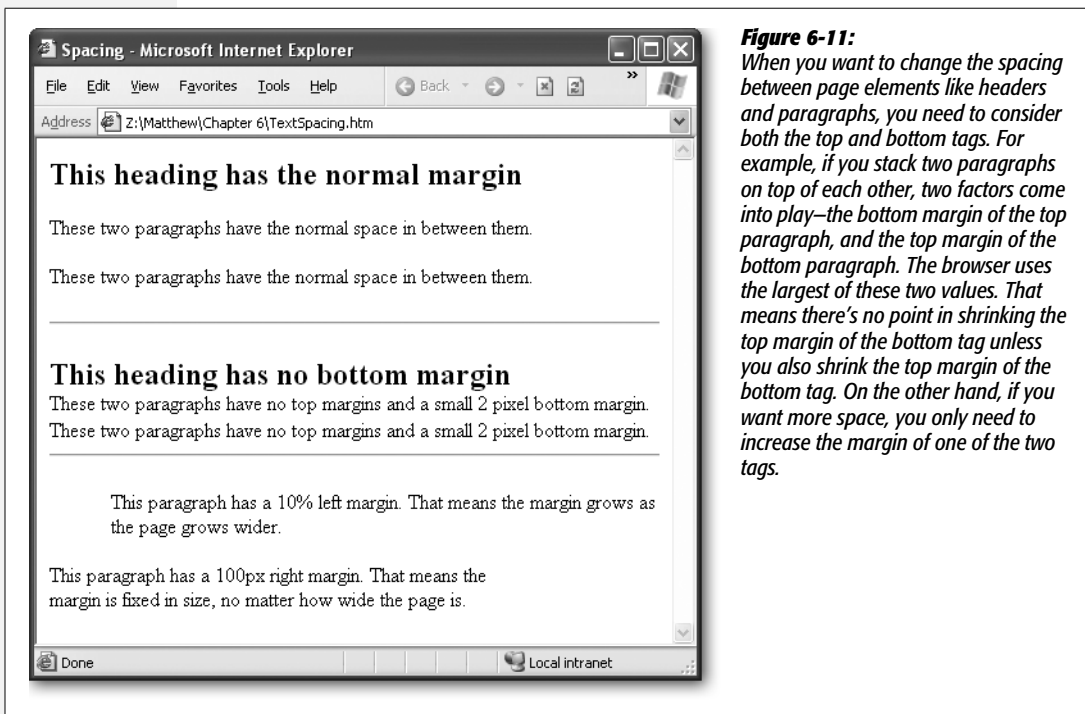
**Tip:** You can supply margins as fixed pixel values (in which case you always get the exact same size) or as percentages (in which case the margin is a percentage of the width or height of the current document window).

This particular rule doesn't have much effect, because eight pixels is the standard margin that Web browsers apply around block elements (on all sides). The eight-pixel margin ensures a basic bit of breathing space. However, if you're looking to create dense pages of information, this space allowance can be a bit too generous. Therefore, many Web site developers look for ways to slim down the margins a little bit.

One common trick is to close the gap between headings and the text that follows them. Here's an example that puts this into action using inline styles:

```
<h2 style="margin-bottom: 0px">This heading has no bottom margin</h2>  
<p style="margin-top: 0px">This paragraph has no top margin.</p>
```

You'll notice that this style rule uses the more targeted *margin-top* and *margin-bottom* properties to home in on just one margin. You can also use *margin-left* and *margin-right* to set different margins on all sides. Figure 6-11 compares some different margin choices.



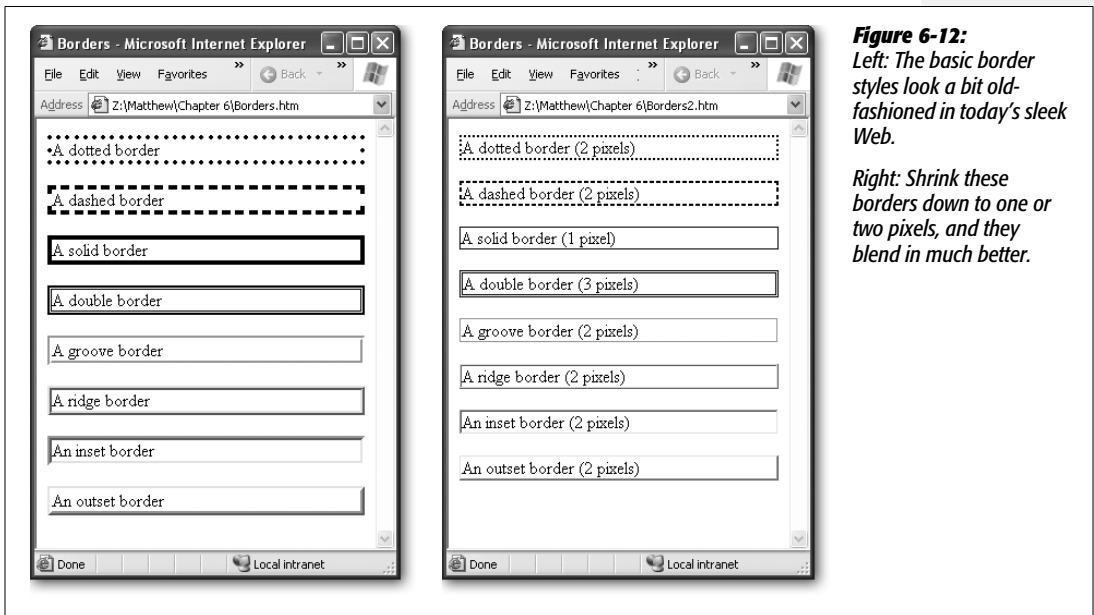
**Figure 6-11:** When you want to change the spacing between page elements like headers and paragraphs, you need to consider both the top and bottom tags. For example, if you stack two paragraphs on top of each other, two factors come into play—the bottom margin of the top paragraph, and the top margin of the bottom paragraph. The browser uses the largest of these two values. That means there's no point in shrinking the top margin of the bottom tag unless you also shrink the top margin of the bottom tag. On the other hand, if you want more space, you only need to increase the margin of one of the two tags.

If you're daring, you can even use *negative* margins. Taken to its extreme, this can cause two tags to overlap.

**Note:** Unlike most other CSS properties, margin settings are never inherited. That means if you change the margins of an element, other elements inside that element aren't affected.

## Borders

The last group of style sheet properties that you'll learn about in this chapter let you add borders to your Web pages (Figure 6-12). Borders are a great way to separate pieces of content. The only thing that's better than borders for organizing information are tables, which you'll learn about in Chapter 9.



**Figure 6-12:** Left: The basic border styles look a bit old-fashioned in today's sleek Web. Right: Shrink these borders down to one or two pixels, and they blend in much better.

Table 6-5 lists the three key border properties.

**Table 6-5. Border Properties**

Property	Description	Common Values	Oldest Supported Browsers	Can Be Inherited
border-width	The border width sets the thickness of the border line. Usually, you'll want to pare this down.	A pixel width.	IE 4, Netscape 4	No

**Table 6-5.** Border Properties (continued)

Property	Description	Common Values	Oldest Supported Browsers	Can Be Inherited
border-style	Browsers have eight built-in border styles. The border style determines what the border line looks like.	none dotted dashed solid double groove ridge inset outset	IE 4, Netscape 4	No
border-color	The color of the border line.	A color name, hexadecimal color code, or RGB value (see page 152).	IE 4, Netscape 6	No

## Basic Borders

The first choice you make when creating a border is the style. Depending on the style you pick, you can add a dashed or dotted line, a groove or a ridge, or just a normal thin hairline (which often looks best). Here's a rule that creates a dashed border:

```
p {
  border-style: dashed;
}
```

To make a border look respectable, you need to reduce the border width. The standard border width is almost always too clunky. You should reduce it to one or two pixels (depending on the style):

```
p {
  border-style: dashed;
  border-width: 2px;
}
```

---

**Tip:** You can also use properties like *border-top-style* and *border-left-width* to set different styles, width, and colors for every side of your element. Using many properties at once can occasionally create an unusual effect, but usually you don't need to get this detailed. Instead, check out the border optimization tips in the next section.

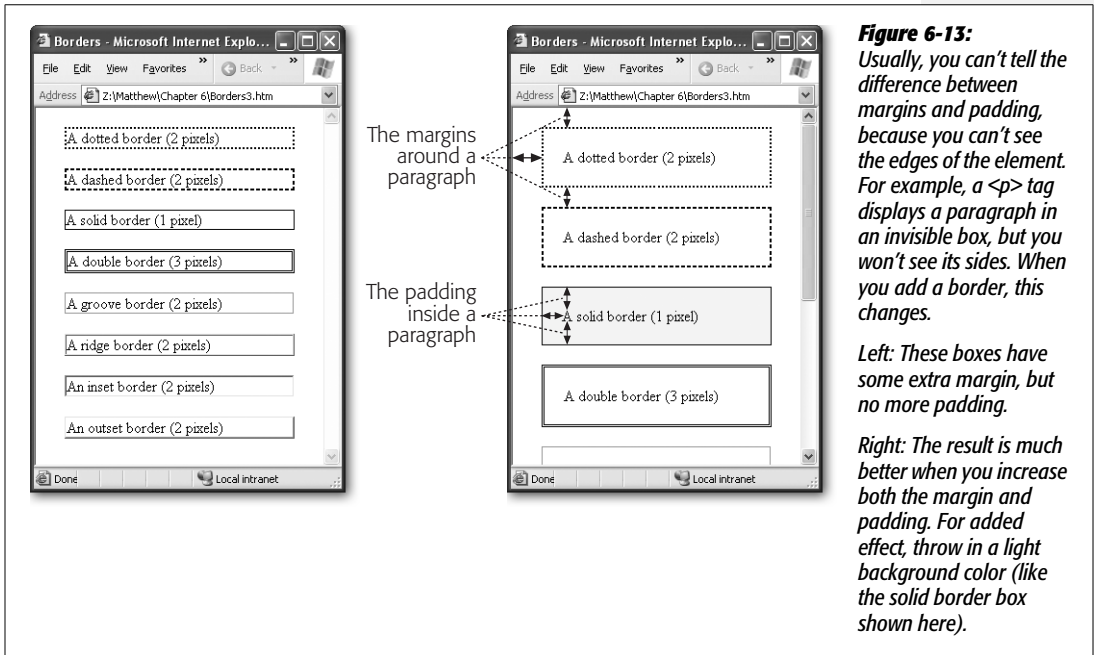
---

## Making Better Borders

In Figure 6-12 the actual borders look fine, but they are squashed too close to the text inside the boxes formed by the borderlines, as well as by the edges of the page.

To make a border stand out, consider using the `border` property in conjunction with three other properties:

- **background-color** (page 151) applies a background color to your element. When used in conjunction with a border, it makes your element look like a floating box.
- **margin** (page 165) lets you set the spacing between your border box and the rest of the page. Increase the margin so that your boxes aren't crowded up against the rest of the content in your page or the sides of the browser window.
- **padding** works like the `margin` property, but it sets spacing *inside* your element, between the invisible edges and the actual content. Increase the padding so that there's a good amount of space between the border and your text. Figure 6-13 shows the difference between margin and padding.



Here's an example of a paragraph that looks like a shaded box:

```
p {
  background-color: #FDF5E6;
  margin: 20px;
  padding: 20px;
  border-style: solid;
  border-width: 1px;
}
```

Figure 6-13 shows how the *margin*, *padding*, and *background-color* properties change an ordinary paragraph into a shaded box.

## Using Borders to Separate Sections

In Chapter 5 (page 121), you learned about the unremarkable `<hr>` tag, which gives you a quick and easy way to separate one section of text from another with a horizontal line. With style sheets, you get several more ways to create attractive separators.

The first line of attack is to style the `<hr>` tag itself. You can use the *width* property to shrink the separator down. You supply length in terms of the percentage of a line's full length. For example, here's a half-length line that's centered on the page:

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

You can also thicken the line by using the *height* property and supplying a thickness in pixels. Here's a thick line:

```
hr {  
  height: 5px;  
}
```

For a variety of more interesting effects, you can bring borders into the mix. For example, here's a rule that heightens the horizontal line, applies the *double border* style, and adopts a modern light gray color:

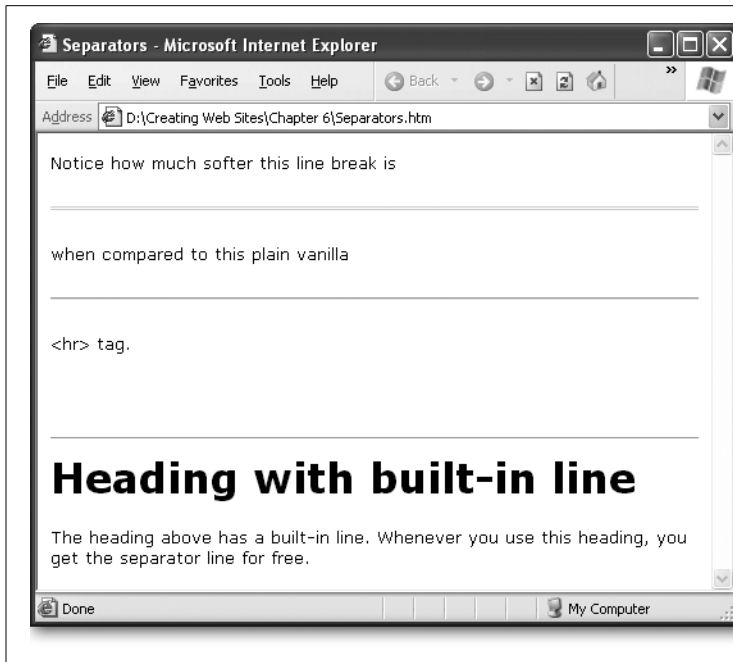
```
hr {  
  height: 3px;  
  border-top-width: 3px;  
  border-top-style: double;  
  border-top-color: #D8D8D8;  
}
```

This gives you a quick way to revitalize all your separators. However, if you aren't already using the `<hr>` tag, you don't need to start now. Another option is to bind the horizontal line to another tag, like a heading. For example, the following `<h1>` tag adds a grooved line at the top. The *margin* property sets the space between the line and previous tag, while the *padding* sets the space between the line and the heading text.

```
h1 {  
  margin-top: 30px;  
  margin-bottom: 20px;  
  padding-top: 10px;  
  border-top-width: 2px;
```

```
border-top-style: groove;
}
```

Figure 6-14 shows both these examples.



**Figure 6-14:**  
This document includes (from top to bottom), a customized `<hr>` line, a normal `<hr>` separator, and a `<h1>` heading with a top border.

## Class Selectors

So far, you've seen how to apply formatting rules on a tag-by-tag basis. These selectors are called *type selectors*. They apply formatting by matching every occurrence of an HTML tag. The only exception is inline styles (page 139), which act only on the tag where they're placed.

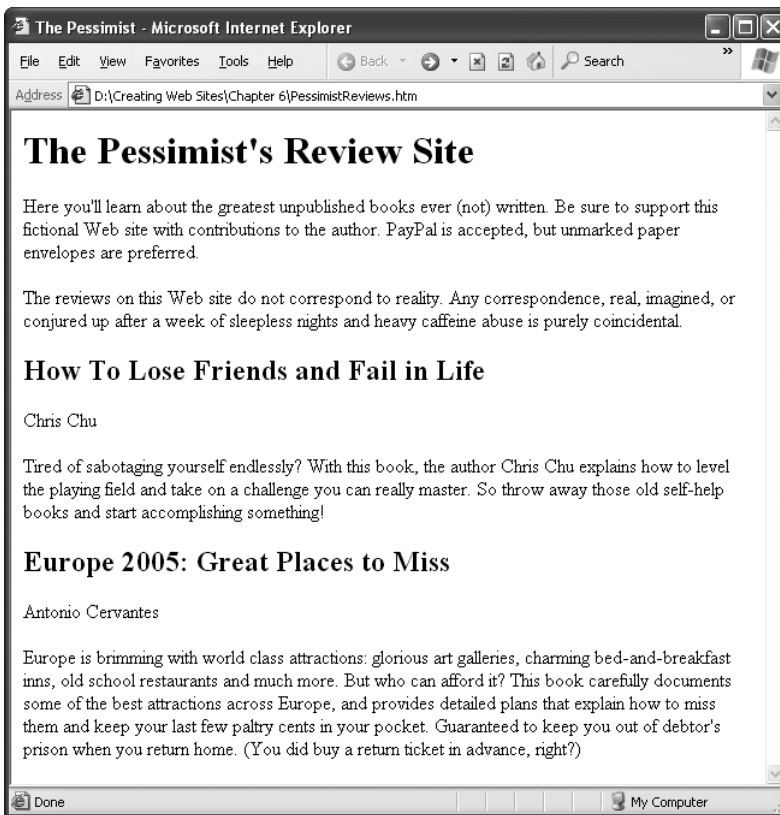
Type selectors are powerful, but not that flexible. Sometimes you need a little more flexibility to modify whole sections or small portions of an HTML document. Fortunately, style sheets have the perfect solution with *class selectors*.

Class selectors are one of the most practical style sheet tricks around. They allow you to separate your rules from your tags, and use them wherever you please. The basic idea is that you separate your Web page content into conceptual groups, or *classes*. Once you've taken this step, you can apply different formatting to each class. The trick is, you choose where you want to use each class in your Web page. For example, you might have two identical `<h1>` headings, but give them separate classes so the formatting is different for each heading.

For a more detailed example, consider the page shown in Figure 6-15. In the following sections, you'll work with this example to apply class-based style rules.

## Creating Class Rules

To use classes, begin by mentally dividing your page into different kinds of content. In this case, it makes sense to create a specialized class for book reviews, and the author byline.



**Figure 6-15:** In the average HTML document, you have a sea of similar tags—even a complex page often boils down to just headings and paragraph tags. This page has a general introduction followed by a series of book reviews. The general introduction, the author credits, and the book summaries are all marked up with `<p>` tags, but they shouldn't be formatted in the same way, because they represent different types of content. Instead, a better system would be one in which each different type of content (title, author, or description) gets formatted in a different way.

To create a class-specific rule, you use a two part name, like this:

```
p.review {
    ...
}
```

The first part of the name indicates the tag that the rule applies to—in this case, the paragraph tag. The second part (the part after the period) is the class name. You can choose whatever class name you want, as long as you stick to letters, digits, and dashes, and make sure the first character is always a letter.

The point of the class name is to provide a succinct description of the type of content you want to format. In this example, the class name is `review`, because it's going to be applied to all the paragraphs that contain the actual reviews.

So how does the browser know when to apply a rule that uses a class selector? It turns out that class rules are never applied automatically. Instead, it's up to you to add the class name to the appropriate tags using the class attribute in your HTML file. Here's an example that links a paragraph to the review class:

```
<p class="review">The actual review would go right here.</p>
```

As long as the class name in the tag matches the class name in the style sheet, the browser applies the formatting.

---

**Note:** Class rules work *in addition* to any other rules. For example, if you create a rule for the `<p>` tag, that rule applies to all paragraphs, including those that are part of a specialized class. However, if the class rule conflicts with any other rules, the class rule wins.

---

Here's the complete style sheet you might use to format the book review page:

```
/* Set the font for the whole page. */
body {
    font-family: Georgia, serif;
}

/* Set some standard margins for paragraphs. */
p
{
    margin-top: 2px;
    margin-bottom: 6px;
}

/* Format the heading with a background color. */
h1 {
    background-color: #FDF5E6;
    padding: 20px;
    text-align: center;
}

/* Make the bylines small and italicized. */
p.byline {
    font-size: 65%;
    font-style: italic;
    border-bottom-style: outset;
    border-bottom-width: 1px;
    margin-bottom: 5px;
}
```

```

    margin-top: 0px;
}

/* Make book reviews a little smaller, and justified. */
p.review {
    font-size: 83%;
    text-align: justify;
}

/* Make the review headings blue. */
h2.review {
    font-size: 100%;
    color: blue;
    margin-bottom: 0px;
}

```

This style sheet includes three type selector rules. The first formats the `<body>` tag, thereby applying the same font to the whole Web page. The second gives every `<p>` tag the same margins, and the third changes the alignment and background color of `<h1>` headings. Next, two new paragraph classes are defined—one for the byline, and one for the review body. Lastly, a class is created for the review headings.

This example also introduces another feature—CSS comments. CSS comments don't look like HTML comments. They always start with the characters `/*` and end with the characters `*/`. Comments allow you to document what each class represents. Without them, it's all too easy to forget what each style rule does in a complicated style sheet.

And here's how the page applies the classes in the style sheet. (To save space, most of the text is left out, but the essential structure is still here.)

```

<html>

<head>
<link rel="stylesheet" href="PessimistReviews1.css"
      type="text/css" >
  <title>The Pessimist</title>
</head>

<body>

  <h1>The Pessimist's Review Site</h1>
  <p>...</p>
  <p>...</p>
  <br>

```

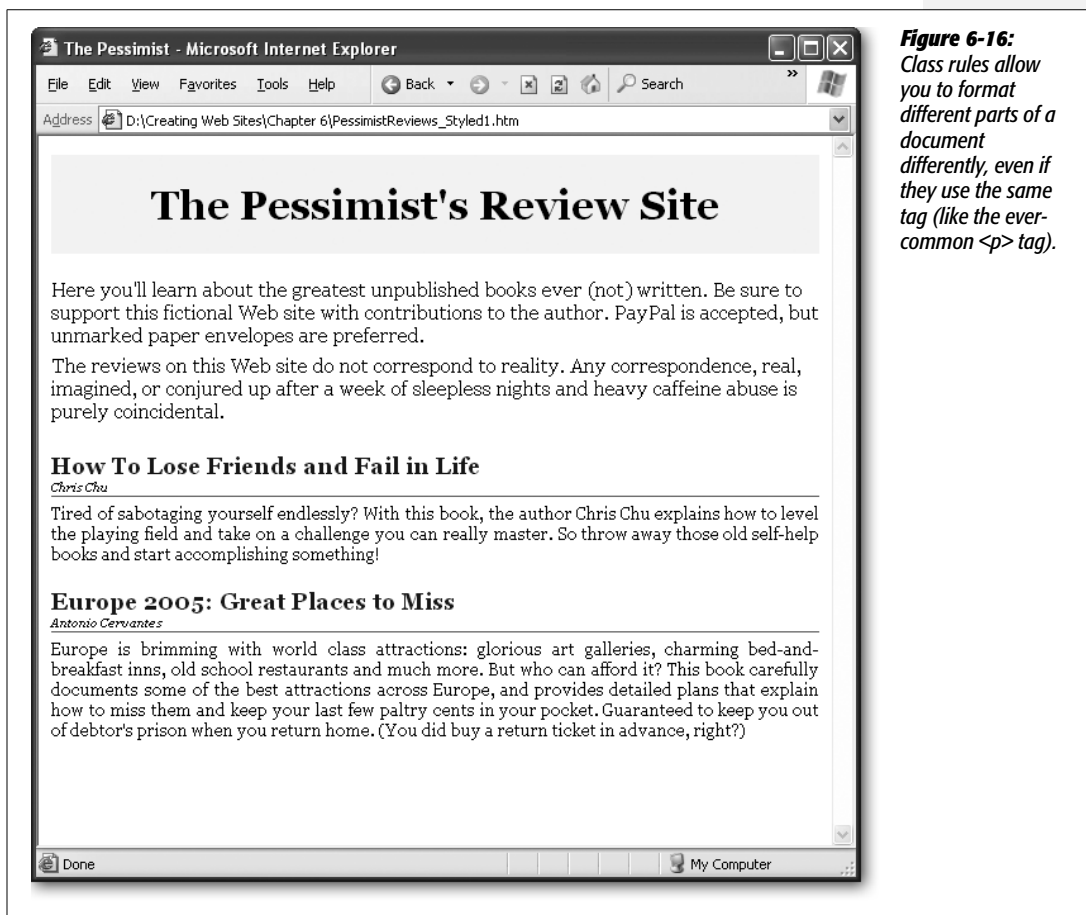
```

<h2 class="review">How To Lose Friends and Fail in Life</h2>
<p class="byline">Chris Chu</p>
<p class="review">...</p>
<h2 class="review">Europe 2005: Great Places to Miss</h2>
<p class="byline">Antonio Cervantes</p>
<p class="review">...</p>

</body>
</html>

```

Figure 6-16 shows the result.



**Figure 6-16:** Class rules allow you to format different parts of a document differently, even if they use the same tag (like the ever-common `<p>` tag).

---

**Tip:** Creating style sheets is an art and takes a fair bit of practice. To make the best use of them, you need to become comfortable with class rules. Not only do class rules give you complete flexibility, they also help you think in a more logical, structured way about your Web site.

---

## Saving Work with the <div> Tag

It can get tedious to apply the class attribute to every tag in your Web page. Fortunately, there's a great shortcut, courtesy of the <div> tag.

You may remember the <div> tag from the last chapter (page 125). It lets you group together arbitrary sections of your Web page. You can put as many elements in the <div> tag as you want, including headings, paragraphs, lists, and more.

Thanks to style sheet inheritance, if you apply a class name to the <div> tag, it's automatically applied to all the nested elements. That means you can change this:

```
<p class="review">...</p>
<p class="review">...</p>
<p class="review">...</p>
```

into this:

```
<div class="review">
  <p>...</p>
  <p>...</p>
  <p>...</p>
</div>
```

This works because of style sheet inheritance (page 148). Essentially, when you format the <div> tag, all the <p> tags inside of it inherit the settings. And although there are some settings that can't be inherited in this way (like margin and padding), most can. Figure 6-17 shows this example.

The <div> tag is a great way to save loads of time. Web gurus use it all the time.

## More Generic Class Rules

You can also create a rule that has a class name but doesn't specify a tag name. All you need to do is leave the first part of the selector (the portion before the period) blank. Here's an example:

```
.emphasize {
  color: red;
  font-weight: bolder;
}
```

The great thing about a rule like this is that you can use it with *any* tag, as long as you use the right class name. In other words, you can use it to format paragraphs, headings, lists, and more with bold, red lettering. The class name reflects this more

general-purpose use. Instead of indicating the type of content, it indicates the type of formatting.



**Figure 6-17:** In this example, each review is wrapped in a `<div>` tag. The `<div>` tag applies a background color and some borders, separating the reviews from the rest of the page. Techniques like these can help organize dense pages with lots of information.

Most Web designers use both tag-specific class rules, and more generic class rules. Although you could stick exclusively with generic rules, if you know that a certain set of formatting options will only be used with a specific tag, it's good to clearly indicate this fact with a tag-specific rule. That way, you won't forget the purpose of your rule when you edit your Web site later on.

## Creating a Style Sheet for Your Entire Web Site

Class rules aren't just useful for separating different types of content. They also come in handy if you want to define the rules for your entire Web site in a single style sheet.

In a typical Web site, you'll have pages or groups of pages that need to be formatted differently. For example, you might have several pages that make up an online photo gallery, another group of pages chronicling your trip to Guadeloupe, and a

separate page with your résumé. Rather than create three style sheets, you can create a single style sheet that handles everything. The trick is to use different class names for each section. In other words, you'll create a résumé class, a trip diary class, and a photo gallery class. Here's a basic outline of this approach:

```
/* Used for the resume pages. */
p.resume { ... }
h1.resume { ... }
h2.resume { ... }
...
/* Used for the trip diary pages. */
p.trip { ... }
h1.trip { ... }
h2.trip { ... }
...
/* Used for the online photo gallery. */
p.gallery { ... }
h1.gallery { ... }
h2.gallery { ... }
...
```

Obviously, each page will use only a few of these rules. However, it's often easier to maintain your site when you keep your styles together in one place.